Departamento de Ingeniería del Software e Inteligencia Artificial

Proyecto Fin de Máster en Sistemas Inteligentes

Máster en Investigación en Informática

Facultad de Informática

Universidad Complutense de Madrid

SUPERCOMPUTING QUALITY SCHEDULING

A SOFT-COMPUTING APPROACH

SERGIO TARANCÓN FAUS

LUIS GARMENDIA SALVADOR
MATILDE SANTOS PEÑAS

Madrid, Septiembre 2009

La vida es lo que te pasa
mientras haces planes

AGRADECIMIENTOS

Esta es quizás, la página que más gusto da escribir después de haber redactado decenas de ellas. Cuando te sientas a ello y miras todo el trabajo hecho, el esfuerzo realizado y sobre todo, a quienes te han acompañado en el camino, no quieres otra cosa que, con lágrimas en los ojos, darles infinitas gracias.

En primer lugar, quiero dar las gracias a mis padres: Papá, Mamá, sin vosotros no estaría aquí; no sería quien soy. Gracias por haberme enseñado tan bien lo que es andar por la vida, por haberme apoyado siempre y por haber confiado siempre en mí. Siento no haber ido más domingos a comer a casa ni haber pasado unos días en la playa este verano, pero bien sabéis vosotros, que tenía que hacerlo. De todo corazón, gracias.

En segundo lugar, a mis hermanas: Ana María y Gema, que son la alegría de la casa; las risas, las sonrisas, los besos, los abrazos, las tonterías y las cosas serias. Estoy muy orgulloso de vosotras, del año que habéis hecho, de todo lo que habéis conseguido con vuestro esfuerzo y de lo que habéis dejado atrás. Espero no tener que decir nunca más - "no puedo: tengo que trabajar en la tesis" - Espero no perderme más minutos de vuestra vida. De verdad, muchas gracias por haber llegado detrás de mí a alegrar mi vida.

En tercer lugar a todo ese batallón de amigos que han sufrido este año mi ausencia: los del fútbol (Nacho, Javi, Luis, Mario,…), los de la facultad (Félix, Frank, ,…), a Bolsón y a Kenobi, a Rober y Angelito, y muy especialmente a Mel por su corrección de última hora (*you know what I mean*), a todos los que mantengo en la agenda y con los que, este año, he compartido mi "no puedo".

Muy especialmente a Toño que estoy seguro es el que más ha sufrido este año de desaparición y espero "poder quedar hoy, mañana y los próximos siete fines de semana". Bien sabe él que no le puedo prometer que no me volveré a "enfangar" ni que "*cuatripatti mi via quedá*" pero espero seguir aprendiendo de mis errores y, al menos, no caer en los mismos. Me siento enormemente afortunado de ser tu amigo, de que ya sean más de diez años los que nos han visto juntos y todos los que faltan por pasar. Gracias por haber soportado este añito que te he dado. Gracias amigo.

Finalmente y no por ello menos importante, quiero agradecer su cariño, empuje y confianza a esa niña de rosa que el destino puso en mi vida camino de Asturias. Sabe perfectamente que sin su apoyo, hoy no estaría escribiendo esto ni cerrando todos los temas que tenía pendientes. SQS ha sido un trabajo de investigación, de prueba y error, de aprendizaje, de "prepucio", de hipótesis y conclusiones, de principales aportaciones y de trabajos futuros; con su introducción, su motivación, su problemática y su planteamiento de una solución inteligente; con un marco teórico y una propuesta práctica. Sara, nadie sabe mejor que tú qué significa SQS. Este es otro de los capítulos del libro que empezamos a escribir a orillas del Sella. Gracias, mi amor por el año vivido, el día de hoy y los que tienen por venir. Gracias por quedarte conmigo.

Madrid, septiembre de 2009

ABSTRACT

In the past few years, Grid computing has become one of the most promising technologies all around this globalized world we live in. This technlogy is being used more and more in all types of industries and in a few years time it will be present in all domestic homes.

With this perspective in mind, business variables are leveraging scheduling techniques for evaluating the user's quality of satisfaction, the optimization of profits from the service provider's point of view and more. Until now, Grid scheduling has never taken into account the patterns of the nature of the job or resouce specialization.

The SQS framework casts light on darkness and proposes a new and smart way of scheduling based on soft-computing technologies. Its vision, far from the classical perspective, keeps away from evolutionary algorithms and centers on neuro-fuzzy clustering of jobs and resources. SQS also presents an innovative way of comparing fuzzy sets and a new defuzzyfier method.

Let us present the SQS way of scheduling Grid resources. It will open the minds of the readers to a new method for solving scheduling problems.

KeyWords: *Grid, Supercomputing, Fuzzy logic, Neural network, Evolutionary algorithm, Neuro-fuzzy network, Supervised learning, Online learning.*

RESUMEN

En los últimos años, los sistemas distribuidos en Grid se han considerado una de las tecnologías con mayor futuro. Poco a poco van calando más y más en la industria y pronto llegarán a ser clave en la futura generación de Internet.

Con una orientación cada vez más hacia la industria y la entrada de nuevas variables de negocio, las técnicas simples de planificación de estos sistemas empiezan a quedarse obsoletas a la hora de evaluar la calidad de satisfacción del usuario y la optimización del beneficio por parte del proveedor. Además, la planificación hasta ahora nunca había tenido en cuenta la naturaleza de las tareas ni la posible especialización de los recursos.

El *framework* SQS trata de solucionar toda esta problemática y propone un método inteligente basado en técnicas de *soft-computing*. Su visión, lejos de las propuestas clásicas basadas en algoritmos genéticos, plantea la clasificación neuro-borrosa de tareas y recursos buscando un patrón común, mucho más sencillo de comparar. Así mismo, SQS presenta un método innovador para la comparación de conjuntos borrosos y de *desborrosificación*.

Palabras clave: Grid, Supercomputación, Lógica difusa, Red neuronal, Algoritmo evolutivo, Red Neuro-borrosa, Aprendizaje supervisado, Aprendizaje en línea.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

PREFACIO

Controlar el tiempo. Ese ha sido, a lo largo de la historia, uno de los mayores retos del ser humano. El hombre ha aprendido a medirlo y a ajustar sus tareas a él, en definitiva, a convivir con él.

Sin embargo, poder controlarlo sigue siendo uno de los temas más apasionantes para la raza humana. Se sigue trabajando en la posibilidad de los saltos temporales, tanto hacia el pasado como al desconocido futuro. Tratando de detenerlo o acelerarlo, con ánimo desesperado por poder controlarlo a nuestro antojo.

Ante dicha imposibilidad, sólo nos queda ajustarnos a él: planificar nuestras actividades en función de ese tiempo caprichoso que, a veces, parece transcurrir tan deprisa y, en otras ocasiones, tan despacio.

De igual forma que en la vida personal, en el mundo empresarial la gestión correcta del tiempo permite conseguir objetivos. Maximizando beneficios y minimizando riesgos alcanzaremos lo que deseamos: obtener un rendimiento económico mayor.

A medida que avanza la historia de la Humanidad, se han sustituido más y más las tareas que antes habían sido realizadas por humanos: primeramente con animales y, finalmente, con máquinas, automatizando el proceso. En esta tecnológica, la mayoría de

dichas tareas se han mecanizado y, hoy en día, se realizan con una eficiencia y en un tiempo de ejecución que, antaño, eran impensables.

No obstante, esta ventaja trae consigo un nuevo reto: gestionar esa enorme fuerza de trabajo en el tiempo. Con la aparición de los sistemas informáticos distribuidos y, más en concreto, gracias a los últimos avances y descubrimientos en el campo de los sistemas distribuidos en Grid, una buena gestión toma, cada día, una mayor relevancia e importancia. Después de todo disponer de mucha comida no sirve de nada si no sabemos cómo planificar su ingesta.

Por ello, la planificación de tareas en sistemas distribuidos en Grid se ha convertido en uno de los temas de investigación más apasionantes de los últimos tiempos. Su objetivo es adaptar a estos sistemas los modelos clásicos de planificación, planteados por la logística. Este reto nos lleva, ineludiblemente, a otro en el que, todos los equipos que pertenezcan al sistema en Grid, puedan trabajar de forma cooperativa. Así se consigue un mayor número de tareas resueltas en un tiempo menor. Con todas estas premisas, se consigue maximizar el beneficio del proveedor minimizando, así mismo, el tiempo de espera del cliente.

Tratando de resolver este problema tan complejo en términos computacionales, este trabajo aboga por el uso de la Inteligencia Artificial. En concreto, se propone el uso de técnicas *soft-computing*: lógica difusa, redes neuronales y algoritmos genéticos. Dada la naturaleza del problema que se plantea aquí, también se comentará la posible aplicación de los sistemas de razonamiento basados en casos (CBR - *Case-Based Reasoning*). Este tema se tratará de forma sucinta, como propuesta para la futura tesis doctoral.

Pasemos, pues, a comprobar cómo estas técnicas de Inteligencia Artificial resuelven la problemática cuando no es posible establecer un método comparativo dado que, de las tareas a ejecutar como de los recursos con los que se cuentan, no se dispone de información suficiente.

INTRODUCCIÓN

1. Motivación

La naturaleza de los actuales sistemas distribuidos en Grid necesitan contemplar la naturaleza dinámica de sus recursos para realizar una planificación adecuada de los trabajos a procesar en función de las características y disponibilidad de los recursos que engloban (Yu, Luo, Chou, Chen, & Zhou, 2007).

Incluso el estándar de facto de infraestructuras Grid: *Globus Tookit 4* (I. Foster, 1998), (Foster, 2005) no dispone a día de hoy de planificador basado en Inteligencia Artificial que utilice al máximo esta naturaleza dinámica para la planificación (Yu, Luo, Chou, Chen, & Zhou, 2007). Se siguen utilizando técnicas de planificación clásicas provenientes principalmente de la investigación operativa que siguen sin resolver, de una forma óptima, el problema.

Es necesario incluir y trabajar en el aspecto dinámico de los planificadores. Para ello, son necesarias técnicas de Inteligencia Artificial con las que obtener información mucho más detallada sobre el estado de los recursos y sus características. De esta forma, se puede obtener una clasificación de qué recursos son más apropiados para qué trabajos. Así mismo, se deben considerar las características de dichos trabajos, evaluando su comportamiento de ejecución sobre los diferentes recursos existentes. Tampoco se han de

olvidar las condiciones en que se ejecutan las tareas, lo que permite inferir qué recursos serán los mejores a la hora de asignar dichas tareas.

En el sentido de incluir técnicas de Inteligencia Artificial, y salvar el hueco existente entre la planificación en tiempo real y la resolución de un problema NP-completo (o duro, dependiendo de lo ambiciosos que seamos), se tratará de aprender de las ejecuciones de tareas determinadas sobre recursos particulares, para así ajustar las futuras asignaciones. Resolviendo, de este modo, el problema de planificación,

En nuestro caso, buscaremos en las técnicas de *soft-computing* (lógica difusa, redes neuronales y algoritmos genéticos) metodologías con las que analizar la ejecución de tareas en recursos dentro de un sistema distribuido en Grid y pronosticar su estado futuro para realizar la planificación.

Toda esta información nos servirá para obtener patrones de cómo se comportan ciertas tareas en determinados recursos. Con ello, se conseguirá inferir más fácilmente qué tarea, trabajo o aplicación se ejecutará mejor en qué recurso.

2. Descripción del problema

En esta sección se va a tratar de presentar casos de la vida real donde la planificación y, más concretamente, la relativa a los sistemas distribuidos en Grid, es de gran importancia y necesidad.

Antonio es un ingeniero aeronáutico que acaba de terminar un modelo de ala de avión con el que, teóricamente, se reducirá el consumo de carburantes a la atmósfera en un 25 %. Antonio ha introducido un número de variables inédito hasta ahora y la estación de trabajo de su empresa no le permite hacer una simulación con tantas variables.

En la otra punta del país, Gema acaba de descubrir una posible solución para el enfisema pulmonar al descubrir una proteína sobre la que, variando su comportamiento, puede reducirse la probabilidad de desarrollar esta enfermedad en un 34 %. Necesita procesar muchos datos en muy poco tiempo. Su proyecto no tiene restricciones económicas al ser clave en la estrategia de I+D de su país.

Por último, la más exitosa *startup* española de desarrollo multimedia acaba de terminar su prototipo de dinámica de fluidos que consigue un realismo en las imágenes 3D con la que parece que las gotas salpican desde la pantalla de cine. Han de demostrar que su aplicación puede funcionar en sistemas en producción reales y que pueden obtenerse los resultados en un tiempo razonable. De esta prueba depende una alianza con el mayor productor de contenidos 3D del mundo.

Todos estos escenarios necesitan, de alguna manera, supercomputación. Algunos tienen restricciones económicas o temporales, otros no.

De igual manera, hay que considerar el punto de vista del proveedor de esa supercomputación. Normalmente serán empresas con grandes *data center* y granjas de ordenadores. Pero, también puede darse el caso de que, organizaciones no tan dotadas, quieran aprovechar los tiempos ociosos de sus máquinas para procesar trabajos, tanto ajenos como propios.

En este escenario, se plantea el problema de tener que realizar una planificación bajo dos puntos de vista: la del cliente y la del proveedor.

Desde el punto de vista del cliente, lo más relevante, como se ha podido ver en los ejemplos anteriores, es que su tarea pueda completarse en un tiempo concreto, bajo un presupuesto razonable. Además, la información que el usuario (no necesariamente relacionado directamente con el mundo de las tecnologías de la información) puede aportar a la hora de enviar un trabajo al sistema planificador no va a ser suficiente.

Desde el punto de vista del proveedor, lo más interesante es que sus recursos estén permanentemente en uso. Además, deseará que terminen las tareas lo más rápidamente posible para poder ocuparlos con otras nuevas. Maximizando su beneficio económico.

Como puede intuirse, la calidad del servicio (*Quality Of Service*) va a ser un elemento clave en ambas perspectivas: serán la base sobre la que se maximicen las estrategias de planificación.

Para planificar las tareas en los recursos, se va a presentar una estrategia novedosa. Va a trabajar con una doble incertidumbre, tanto la de las características de las tareas como la del estado de los recursos en cada momento.

Por un lado, se pretende clasificar las tareas a partir de datos que, a priori, van a estar ocultos en su propia naturaleza. De ellas tan sólo se conocen datos relativos de su tamaño, de su número de bucles, de su entorno de ejecución necesario,… Características que se utilizarán como variables de entrada dentro del sistema de clasificación. Ya en (Schopf, 2006) se cita la problemática que los sistemas distribuidos en Grid presenta a la hora de analizar las necesidades de las tareas ya que no proporcionan datos suficientes o los datos dependen muy y mucho de las características de los recursos donde se ejecutan.

Por otro lado, el estado de cada recurso podemos conocerlo en un momento determinado en función de su uso de CPU, la cantidad de memoria RAM utilizada o el ancho de banda del que dispone. Todas estas características se utilizarán como variables de entrada para la clasificación de los recursos dentro de nuestro sistema clasificador.

Nuestra hipótesis va a considerar que, dada una tarea ejecutada sobre dos recursos distintos, siendo uno con mejores características que el segundo, no va a generar necesariamente mejor rendimiento de ejecución en el primero que en el segundo. Además, la característica más notoria de los sistemas distribuidos en Grid, es su heterogeneidad. No sólo consideraremos que cada recurso será distinto en características del resto, sino que, incluso dos recursos exactamente iguales, no se encontrarán necesariamente en el mismo estado en un momento dado, en términos de porcentaje de uso, de velocidad de sus procesadores, de cantidad de memoria RAM o de la velocidad de la red. Todos estos valores, ahora, se vuelven imprecisos y la incertidumbre vuelve a reinar en el sistema.

3. Objetivos

El objetivo de este trabajo es considerar la incertidumbre intrínseca que existe en la definición de las características de una tarea. En realidad, dichas características sólo se conocen, explícita y únicamente, cuando ya se ha ejecutado sobre un determinado recurso. Así mismo, también se va a considerar una clasificación de los recursos

introduciendo incertidumbre a su monitorización en tiempo real para hacer así que el ajuste entre tareas y recursos se dé en términos difusos en vez de en términos nítidos.

Para conseguir este tipo de clasificación, se va a trabajar con una red neuro-difusa en la que se van a codificar reglas definidas por un experto. Estas reglas indicarán qué clase de recursos será la mejor opción para la ejecución de una determinada clase de tareas.

Una vez la red neuro-difusa indique que una determinada tarea pertenece a una categoría concreta y se decida cuál es el mejor recurso para ella, se lanzará a ejecución y se monitorizarán los resultados obtenidos. Estos resultados servirán para que nuestro sistema aprenda. El fundamento de este aprendizaje es disponer de entradas (características de tareas y recursos) y de salidas (resultados de la ejecución). Estos datos, junto con las reglas de negocio con las que optimizar la satisfacción de usuarios y clientes, nos proporcionarán el patrón de ejecución. El intensivo uso del sistema proporcionará los datos clave para el entrenamiento de la red neuro-difusa.

En el caso de este sistema, sistema experto difuso y red neuronal se hibridarán dando origen a un sistema neuro-difuso en el que las reglas del sistema experto se codificarán en ciertas neuronas de la red neuronal y el aprendizaje de la red neuronal recaerá sobre el peso de las uniones sinápticas dando más importancia a unas reglas y a otras y perfeccionando y ajustando el conocimiento a priori del experto en función de la experiencia.

Como último objetivo, se desea conseguir un lenguaje de definición de reglas de negocio que puedan ser definidas por el cliente y por el proveedor y que sirvan como reglas para optimizar todavía más la QoS y el beneficio para el proveedor. Para esta tarea se pretende utilizar como base un lenguaje estándar como es JSDL y para los recursos RDL cuya mayor ventaja es su extendida aceptación y uso por la mayor parte de gestores de recursos de sistemas distribuidos en Grid.

4. Estructura del documento

La metodología utilizada en este trabajo se puede dividir en las siguientes etapas.

En la primera parte, se ha realizado una fase de investigación de la literatura y trabajos disponibles al respecto de la teoría de planificación con la que se ha asentado la problemática a resolver y las distintas versiones que se pueden considerar de este problema. Se ha pormenorizado más en el campo de la planificación de tareas sobre recursos computacionales y se ha ido progresando hacia los sistemas heterogéneos, así como hacia los distintos escenarios donde es interesante aplicar planificación. También se han revisado las técnicas de Inteligencia Artificial más comunes a la hora de resolver este problema y se ha hallado un nicho en el que la hipótesis planteada tiene cabida.

En la segunda parte, se ha propuesto una arquitectura para el planificador inteligente basado en *soft-computing*. Para ello, nos apoyaremos en *framework* especializados en lógica difusa, redes neuronales y *middleware* Grid con los que justificar nuestra propuesta. Aquí se verá el punto de vista que SQS tiene de la planificación: su base en la clasificación utilizando redes neuro-difusas, su algoritmo de toma de decisiones basado en valores borrosos y el método de aprendizaje para las redes neuro-difusas con las que mejorar la toma de decisiones de las mismas.

Con todo ello, se presentarán las conclusiones obtenidas y propondremos líneas de trabajo futuras con las que enriquecer el trabajo realizado en esta investigación.

1. ESTADO DEL ARTE

1. El problema de planificación y los sistemas distribuidos en Grid

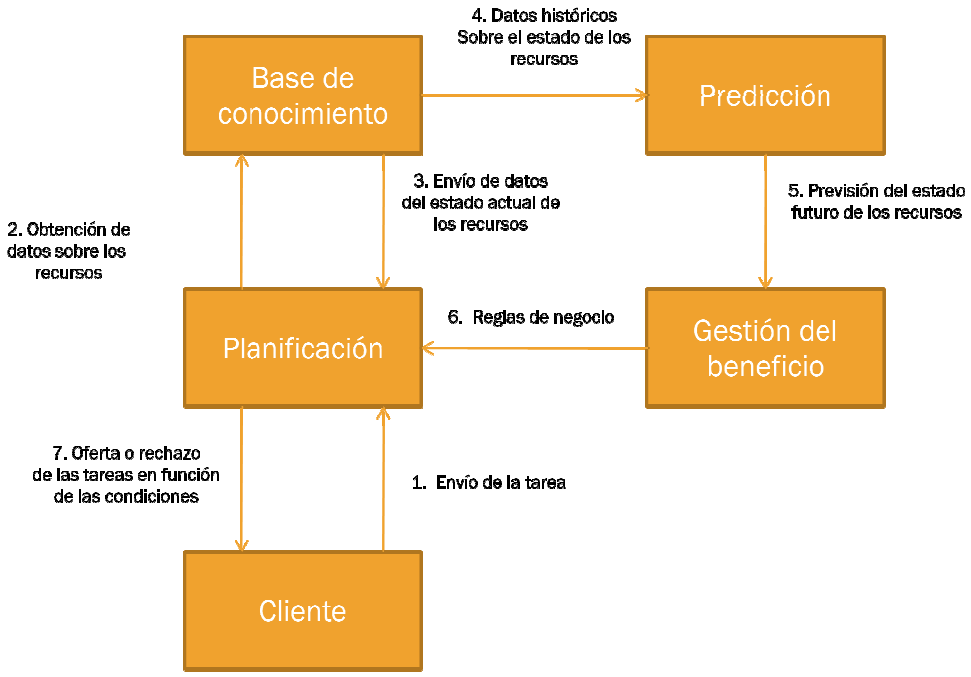
ORÍGENES DEL PROBLEMA DE PLANIFICACIÓN

Los orígenes del problema de planificación provienen de la Investigación Operativa donde se han investigado técnicas de planificación que poder aplicar a problemas de logística, organización, fabricación, ingeniería…

> *La planificación es un proceso de toma de decisiones que se utiliza regularmente en muchas industrias de logística y de servicios. Consiste en asignar tareas a recursos durante espacios de tiempo y su objetivo es optimizar uno o más objetivos. (Pinedo, 2002)*

Hablar de planificación, generalmente nos lleva a pensar en la que se realiza en sistemas logísticos o cadenas de producción industriales. En estos contextos, cada tarea tiene que realizarse en un recurso determinado bajo un orden específico y bajo ciertas condiciones que definen, tanto la tarea, como el recurso.

De igual manera, puede hacerse una distinción en cuanto a la planificación orientada a la Gestión de proyectos (*Project Management*), y a la planificación de recursos y tareas (*Scheduling*). Será esta última el objeto de nuestro interés.



icios.

No obstante, la planificación que a nosotros nos interesa en este estudio tiene que ver más con la que (Pinedo, 2002) denomina como "Planificación de servicios". Este tipo de planificación tiene que lidiar con problemas como la reserva de recursos, tener que trabajar con distintas funciones de toma de decisiones, apoyarse en los datos obtenidos de experiencias pasadas o predecir cual será su comportamiento futuro. La figura 1.1 representa este tipo de planificación:

En la figura 1.1 puede verse cómo el cliente va a enviar el trabajo que quiere realizar al planificador. Éste va a solicitar a la base de conocimiento los datos sobre los recursos que están disponibles en ese momento junto con su disponibilidad. A su vez, los datos históricos sobre el estado de los recursos pasan al módulo de predicción que pronostica el comportamiento que va a tener la ejecución de la tarea. Esos datos pasan al

módulo donde se encuentran las reglas de negocio con las que se calcula el beneficio que se obtendrá de ejecutar esa tarea en ese recurso seleccionado y se pasan esos resultados finalmente al planificador. El planificador, envía la oferta de ejecución al cliente o bien rechaza ejecutar la tarea en las condiciones solicitadas por el cliente y, dependiendo de dicha decisión, se ejecuta finalmente la tarea o no.

El problema a resolver pues consiste en asignar tareas a recursos durante un tiempo determinado de la mejor manera posible. A ser posible, de forma óptima.

La modelización del problema de planificación más extendida es la propuesta por (Pinedo, 2002) que consiste en:

Se desea realizar *n* tareas *Jj (j = 1, 2,…, n)* y se dispone de *m* recursos *Mi (i = 1, 2,…, m)*. Se va a asumir que cada recurso no puede realizar trabajos simultáneamente y que, para cada instante dado, se estará ejecutando únicamente un trabajo en un recurso (hipótesis de no simultaneidad).

Cada trabajo *Jj* representa los datos y variables asociados de la figura 1.2.

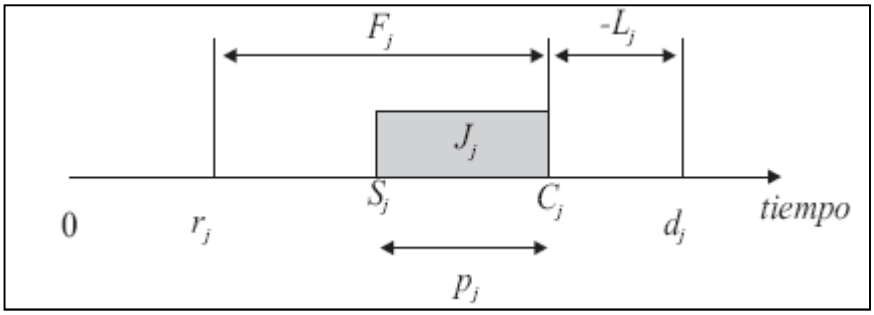


Figura 1.2. *Datos y variables asociados a un trabajo.*

La clasificación triparamétrica consta de tres variables α|β|γ, que representan la siguiente información:

- **α:** Características de los *m* recursos M_i *(i = 1,…, m).*
- **β:** Características de las *n* tareas a procesar J_j *(j = 1,…, n).*

- **γ:** Criterios y modo de optimización que se van a utilizar.

  Datos y variables asociados a un trabajo:

  - Uno ó más **tiempos de procesamiento** Pj_i *(i = 1,…, m)* necesarios para procesar el trabajo *Jj* en cualquier recurso *Mi (i = 1,…, m)*.
  - Una **fecha de disponibilidad** (*rj)*, a partir de la cual puede procesarse dicho trabajo. Si *rj = 0*, para todo *j*, todos los trabajos están disponibles desde el mismo instante de tiempo, estaremos ante problemas de planificación estáticos; mientras que, en caso de distintas fechas de disponibilidad, nos encontraremos ante problemas de planificación dinámicos.
  - Una **fecha de comienzo** (*Sj)*, que indica el instante de tiempo en el que comienza el procesamiento del trabajo *Jj*. No es constante, sino que depende de la planificación.
  - Una **fecha límite, de vencimiento o *due date*** (d_j), en la que el trabajo J_j debería estar terminado para no incurrir en tardanza. Si dicha fecha debe cumplirse estrictamente para obtener factibilidad la denominamos fecha límite (*deadline*).
  - Un **peso o ponderación** (ω_{kj}), que indica la importancia relativa de J_j con respecto a los otros trabajos según el criterio *k* ; tal que $1 \leq k \leq K$, siendo *K* el número de criterios considerados.
  - Una **función de costo real** (f_{kj}) por cada criterio *k*, donde f_{kj} *(t)* es el coste asociado, según el criterio *k*, al trabajo J_j cuando éste se completa en el instante de tiempo *t*. Dicha función dependerá, en general, de los parámetros anteriores.

Dependiendo de si todas las variables anteriores son conocidas antes de ejecutar el algoritmo de planificación, nos encontraremos ante una planificación de tipo determinista (cuando las conocemos) o estocástica (cuando no las conocemos).

Dentro del problema de planificación no sólo se han de atender las características de los trabajos sino también de los recursos.

Puede ocurrir que se disponga únicamente de una máquina ($\alpha = 1$) o de varias y el planteamiento de la planificación será totalmente diferente. Hay que contar con que, al disponer de distintas máquinas, se podrán realizar tareas en paralelo siempre que sean independientes. Igualmente, puede darse el caso de que los recursos estén totalmente especializados y sólo puedan realizar determinados tipos de tareas. En tal situación, no podremos asignar cualquier tipo de tarea a dicho recurso.

En función del tipo de recursos disponibles, podemos clasificar el problema de planificación en:

- **Planificación con recursos especializados**: Cuando los recursos ejecutan siempre las mismas tareas.
- **Planificación con recursos idénticos**, cuando los recursos tienen exactamente las mismas características en cuanto a potencia y velocidad.
- **Planificación con recursos uniformes**, cuando los recursos tienen distinta potencia y velocidad pero son constantes y no dependen de la naturaleza de los trabajos. Esto quiere decir que si un recurso es doblemente potente, la ejecución de cualquier tarea será doblemente rápida sobre ese recurso, independientemente de la tarea que se ejecute.
- **Planificación con recursos no relacionados**, cuando la velocidad de ejecución de la tarea depende tanto de recursos como de tareas y no se puede relacionar la potencia de los recursos.
- **Planificación con recursos no especializados**, cuando existe especialización en los recursos y van asociados a trabajos que se pueden dividir en distintas tareas y que se realizan cada una de esas tareas de forma especializada en el recurso correspondiente. Estos problemas se dividen a su vez en:
    - **Planificación de sistemas *flow shop***, en los que cada trabajo se procesa por todos o algunos de los recursos siguiendo un orden prefijado o un

patrón común. Aquí el orden es fundamental y siempre se sigue el mismo orden o la misma trayectoria por todos los recursos. Este tipo de planificación recuerda a las cadenas de montaje aunque hay que destacar ciertas diferencias:

- Esta planificación puede ocuparse de **distintos tipos de productos**, mientras que en la cadena de montaje sólo se produce un elemento estándar.
- Esta planificación permite el que no se utilicen todos los recursos y **se pueda prescindir de alguno**.
- Esta planificación **no obliga a que exista una dependencia** de que un recurso termine para que otro ejecute una determinada tarea.
- El **tiempo de procesamiento** de una tarea en un recurso puede ser **variable**, mientras que en una cadena de montaje es fijo.

- **Planificación de sistemas *open shop***, todos los trabajos se procesan en todos los recursos y puede realizarse en cualquier orden. En este caso, no se habla de cadena, sino de conjunto de operaciones que se realizan en una determinada máquina en un determinado tiempo. Su principal característica es que el orden es totalmente irrelevante.
- **Planificación de sistemas *job shop***, cuando el subconjunto de recursos que procesa un trabajo y su orden de procesamiento son arbitrarios pero conocidos con antelación. Se puede considerar la planificación *flow-shop* como un caso particular de este en el que todos los trabajos siguen el mismo patrón de flujo.

Las **características de los trabajos** también son determinantes dentro del problema de planificación. Entre las características más notables encontramos:

- **Interrupciones**, es decir, si se permite que la ejecución del trabajo pueda suspenderse y más adelante pueda continuarse donde se había dejado.

- **Relaciones de dependencia entre los trabajos** que condicione el que la ejecución de un trabajo dependa de que otro termine. Estas relaciones suelen representarse mediante un grafo dirigido de tipo acíclico.
- **Fechas de disponibilidad**, que distingan entre los problemas **estáticos** (todos los trabajos comienzan al tiempo) o **dinámicos**.
- **Límite en el número de operaciones que se puedan realizar**, como puede darse en el caso de que existan más tareas a realizar que recursos y conviene establecer un límite superior en la planificación.
- **Tiempo de procesamiento**.
- Posibilidad de utilizar **recursos adicionales**.

El modelo de planificación también tiene que ver con los criterios y las funciones objetivo que se van a considerar, sus características y, en el caso de que exista más de un criterio, qué optimización se va a tratar de considerar: puntos eficientes, puntos extremos, optimización simultánea o jerárquica. En este caso, podemos contar con las siguientes métricas:

- **Tiempo en completar la tarea** J_j desde la fecha de entrega de la misma (C_j)
- La **demora** $L_j = C_j - d_j$. Un valor positivo indica que el trabajo se ha retrasado más de lo previsto, un valor negativo, indica que se ha terminado anticipadamente.
- La **tardanza,** que viene dada por $T_j = max\{0, L_j\}$
- El valor **trabajo retrasado** (U_j), será *1* si el trabajo no se concluye antes de su *due time* y *0* en cualquier otro caso.

Todas estas variables influyen en la definición de las funciones objetivo a minimizar. Suelen tratar de optimizarse el coste máximo o el coste total.

Se define pues un criterio γ_k con $1 \leq k \leq K$. Dicha función, será, en general, dependiente de los tiempos necesarios para completar las tareas. Dependiendo del tipo de problema, se considera incorporar el tiempo en que el recurso está ocioso o no.

Las funciones a optimizar suelen ser de tipo máximo o de tipo suma teniendo en cuenta el coste asociado utilizando el criterio *k* cuando el trabajo *j* se termina en el instante de tiempo C_j.

2. Distintos enfoques del problema de planificación

Según el estudio realizado en (LaTorre, Peña, Robles, & de Miguel, 2008), existen un subconjunto de problemas clásicos que pueden aplicarse a supercomputadores.

EL PROBLEMA DE PLANIFICACIÓN FLOW-SHOP (FSS)

En (Ali, Siegel, Maheswaran, & Hensgen, 2000) se hace un especial análisis de este método. Se trata de un problema *n/m/Cmax* en el que se han de planificar *n* trabajos en *m* máquinas y se trata de que el tiempo global (*makespan*) sea mínimo asumiendo que no se conoce la duración de los trabajos con antelación.

Para este tipo de problemas, se ha encontrado que los mejores algoritmos para resolver este problema es la búsqueda tabú, y los algoritmos genéticos, aunque también se han trabajado con el enfriamiento simulado y las colonias de hormigas.

EL PROBLEMA DE PLANIFICACIÓN JOB-SHOP (JSS)

Consiste en que *n* trabajos tienen que ejecutarse en ciertas máquinas de entre las *m* disponibles.

Los mejores resultados se han obtenido con búsqueda tabú y con algoritmos genéticos. Hay varios artículos a los que hace referencia donde encontrar más detalles.

Según (Wang, Yuan, & Liu, 2008), las restricciones del problema JSS son:

- Cada proceso se debe realizar en la máquina asignada en un determinado momento.
- Cada proceso ha de terminar antes de que empiece el siguiente (secuencialidad)

- Cada parte del proceso se ha de realizar en una máquina en un determinado instante. No se pueden realizar otras partes del proceso en la misma máquina (relación 1-a-1)

Estudios como el realizado en (Wang, Yuan, & Liu, 2008) han utilizado la lógica difusa, y la han aplicado al problema JSS, con el objetivo de maximizar la satisfacción de los clientes. En esta línea, las investigaciones para optimizar este caso particular de optimización se han centrado en fijar variables como el tiempo en construir una unidad o saber cuándo tiene que estar terminada para así optimizar el uso de los recursos. Este tipo de planificaciones están íntimamente relacionados con la planificación de *workflow* que aparece cuando se trabaja con tareas interdependientes.

PROBLEMA DE PLANIFICACIÓN PARA MULTIPROCESADORES (MPS)

Su fundamento es que cada trabajo se puede descomponer en un conjunto de tareas más pequeñas y sus correspondientes dependencias (formando un Grafo Directo Acíclico). La cuestión a resolver es cómo planificar las tareas en los distintos recursos de forma que se cumpla las restricciones de dependencia.

En este tipo de problemas, los algoritmos genéticos han sido los que mejor solución han dado siguiendo dos estrategias: (a) aplicándolos en combinación con otras técnicas de planificación y (b) utilizándolos para mejorar el orden de las tareas dentro una vez están asignadas al recurso.

Otro tipo de problema de planificación es el caso de (Kamer, Bora, & Cevdet, 2008) cuando se desean asignar trabajos a recursos heterogéneos pero que comparten archivos y donde se trata de minimizar el valor de *timearound*. En este caso, el problema es de tipo NP-completo.

Según (Lay, 2003), cuando a parte de existir dependencias entre los trabajos, tareas o aplicaciones, las hay también entre los recursos, el problema pasa de ser NP-completo a NP-duro.

3. Sistemas distribuidos en Grid

BREVE HISTORIA SOBRE LOS SISTEMAS DISTRIBUIDOS EN GRID

Dentro del sitio Web de la Gridipedia (BeInGRID, 2008), nos encontramos con una interesante historia que resume los orígenes de la computación distribuida en Grid.

Los padres del término Grid fueron Ian Foster y Carl Kesselman al querer indicar que la potencia de computación será utilizada de la misma forma que hoy en día utilizamos las redes eléctricas. Sólo tendremos que conectar el equipo a la pared y listo.

Este término no comenzó a utilizarse hasta finales de los años noventa, pero ya mucho antes los clúster Beowulf o software de aprovechamiento de CPU guardaban en su naturaleza este concepto.

En el caso del primero, se convirtió en el primer clúster de bajo coste al utilizar PC estándar como componentes para formar un supercomputador capaz de realizar tareas de alto rendimiento computacional paralelas y estrechamente acopladas. Este primer diseño se utilizó para la NASA, pero en la actualidad todavía se siguen utilizando en entornos científicos.

Esta idea para aprovechar ciclos de CPU ociosos pretendía dividir los trabajos en trabajos más pequeños (tareas) que pudiesen ejecutarse de forma paralela y distribuida dentro de una red de equipos cuando estuviesen ociosos. El proyecto más famoso que seguía esta filosofía fue *Seti@home* que utilizaba los ciclos de reloj de millones de usuarios en todo el mundo para buscar señales de vida extraterrestre.

El primer proyecto que se centró en la integración del software y la administración del mismo fue I-WAY (*Information Wide-Area Year*). Este proyecto, liderado por Ian Foster consistió en conectar 17 centros de súper computación, 5 laboratorios de realidad virtual y 60 grupos de aplicaciones y sirvió como campo de pruebas para aplicaciones avanzadas. Este proyecto fue la base para el desarrollo del que se convertiría en el estándar *de facto* Globus Toolkit (Foster, 2005) con el que se pueden desplegar aplicaciones en una red de recursos heterogéneos.

En 2002, camino a la estandarización de los sistemas distribuidos en Grid, Ian Foster, Carl Kesselman, Jeffrey M. Nick y Steven Tuecke propusieron en su artículo *The*

physiology of the Grid una arquitectura que definía las funcionalidades y comportamientos que debería tener todo sistema distribuido en Grid.

En paralelo con la evolución de la computación en Grid, ha habido intentos por definir servicios Web para interactuar con este tipo de sistemas. Como resultado de todos estos esfuerzos, en 2004, la organización OASIS presentó WSRF (*Web Services Resource Framework*) que consiste en un conjunto de especificaciones de servicios Web para poder implementar funcionalidades OGSA utilizando servicios Web.

FUNDAMENTOS DE SISTEMAS DISTRIBUIDOS EN GRID

Los sistemas distribuidos en Grid son sistemas de computación distribuida que permiten hacer interactuar recursos descentralizados geográficamente para resolver problemas de gran escala (Wikipedia, 2009)

Suelen equipararse erróneamente con los clúster, pero existe entre ellos una gran diferencia en cuanto a que los recursos de los clúster son homogéneos, mientras que los de los sistemas distribuidos en Grid son heterogéneos (Yu, Luo, Chou, Chen, & Zhou, 2007).

Según (Xhafa & Abraham, 2008), los sistemas distribuidos en Grid, cada día forman más parte de la vida real. A continuación se enumeran casos de uso reales en los que se vislumbra futuro para los sistemas distribuidos en Grid:

- Un investigador en computación que busca la manera de hacer óptimo su problema NP-duro favorito en tan solo unas horas.
- Un investigador químico que pretende obtener un nuevo diseño innovador para un nuevo fármaco.
- Un investigador en biomedicina que pretende descubrir la secuencia del ADN de un ser vivo y utilizarla para curar enfermedades.
- Un centro de predicción meteorológica que quiere predecir el acontecimiento de tsunamis.

- Un equipo médico que quiere realizar una operación de cirugía compleja de forma remota utilizando laboratorios virtuales.
- Un economista que desea analizar en tiempo real los valores de su cartera de clientes
- Un estudiante de una universidad a distancia que quiere participar con su equipo a la infraestructura tecnológica de la universidad y trabajar de forma remota junto con el resto de componentes de su clase para conseguir sus objetivos académicos.

LOS DIFERENTES TIPOS DE GRID

El estudio realizado en (Xhafa & Abraham, 2008) nos proporciona una clasificación apropiada sobre los diferentes tipos de Grid con que nos podemos encontrar. A continuación se detallan cuales son estos tipos y una descripción general de sus características. Se va a utilizar la clasificación de la Gridipedia (BeInGRID, 2008) que divide a estos sistemas en tres grandes grupos: (a) por el tipo de recursos que comparten, (b) por la forma en que están distribuidos y (c) por el tipo de servicios que proporciona.

Clasificación en función del tipo de recursos

- **Grid computacionales:** Principalmente comparten recursos de CPU. Este tipo de Grid fue en el que se pensaba al compararlo como las redes eléctricas y donde se han desarrollado las teorías sobre *Utility computing*. Ejemplos de este tipo de Grid son *TeraGrid* y *SETI@home*.
- **Grid de datos:** Su objetivo es compartir recursos de datos, como pueden ser resultados de experimentos, pruebas clínicas… entre los distintos usuarios. Ejemplos de este tipo de Grid son *QCDGrid* o *LHC Computing Grid*.
- **Grid de almacenamiento:** Su objetivo es proporcionar a sus usuarios cantidad de almacenamiento prácticamente ilimitado. El ejemplo más conocido de esto es *Amazon S3*.
- **Grid de equipamiento:** Su objetivo es compartir recursos físicos, como puede ser el caso de grandes telescopios como en el caso del proyecto *eSTAR*.

Clasificación en función de la distribución geográfica de sus recursos

- **Distribuidos a través de Internet:** El objetivo de este tipo de Grid es incluir a todos los que puedan conectarse a Internet. El caso de *SETI@home* y *World Community Grid* son dos de los casos más representativos.
- **Delimitados por Organizaciones Virtuales:** Su objetivo es agrupar distintos recursos contenidos en diferentes organizaciones (académicas, empresariales…) de forma que puedan establecerse sinergias entre ellas. La mayoría de los Grid caen dentro de esta categoría ya que esta cualidad es clave en el diseño de estos sistemas distribuidos.
- **Implementados de forma local:** Normalmente tienen una arquitectura de clúster y suelen ser granjas de máquinas dedicadas únicamente a una tarea. No obstante, existen ya iniciativas para conseguir construir Grid con las estaciones de trabajo de los empleados de las compañías de forma que puedan utilizarse para esas tareas de supercomputación en sus tiempos muertos, durante la noche…

Clasificación en función de los servicios que proporcionan

En la actualidad, el tipo de aplicaciones que se utilizan en sistemas distribuidos en Grid está bastante segmentado principalmente en tres grandes grupos: (a) Dibujo de imágenes, (b) Simulaciones científicas y (c) Aplicaciones Web.

Dentro de esta orientación hacia aplicaciones, cabe destacar la nueva tendencia del mundo Grid hacia el *Cloud Computing* donde se unen las tecnologías Grid con las de virtualización, consiguiendo una nube de recursos virtualizados que se ajustan mucho más a las necesidades de las tareas que se han de ejecutar.

Grids que aprovechan los excedentes de los recursos

Este tipo de infraestructuras aprovechan el tiempo ocioso de los recursos informando al planificador (que estos casos se conoce como *profiler*) para que le envíen la siguiente tarea que esté pendiente de ejecución y de la que pueda encargarse. El mayor problema que tiene utilizar los tiempos ociosos de un recurso es que el resto de procesos

privados que esté ejecutando falsea los datos de disponibilidad y de tiempos para finalizar la ejecución.

Grids para la eCiencia

Este tipo de sistemas distribuidos en Grid se orienta principalmente a la resolución de problemas de ciencia y de ingeniería. En ellos se comparte no sólo potencia de cálculo sino toda la infraestructura, incluidos datos y demás recursos físicos.

Grids de datos

Este tipo de sistemas distribuidos en Grid están orientados principalmente a trabajar, almacenar y administrar grandes cantidades de datos altamente distribuidos. Los temas de investigación clave de esta clase son las políticas de acceso seguras, la replicación de los datos y el movimiento de grandes cantidades de datos.

Grids empresariales

Motivados por la computación de alto rendimiento, las posibilidades empresariales que ofrecen los sistemas distribuidos en Grid son cada vez mayores. Este tipo de Grid permite que se desarrollen distintos proyectos en distintas partes de la empresa o que distintos departamentos compartan sus recursos de una manera transparente. Entre los ejemplos más conocidos encontramos *Sun Grid Engine*, *IBM Grid*, *Oracle Grid* y *HP Grid*.

Desktop Grids

Según (EunJoung, SungJin, HongSoo, & ChongSun, 2008) este tipo de Grid aparece como alternativa de bajo coste a los grandes supercomputadores o clúster. Su intención es interconectar todos los equipos de escritorio que actualmente se encuentran conectados a Internet de forma que pueda aprovecharse al máximo tan tremendo potencial de cálculo. Su mayor problema es la volatilidad de los recursos ya que, al no ser equipos dedicados completamente a procesar trabajos del Grid (pueden estar ocupados con una hoja de cálculo, un procesador de textos o un navegador de Internet, por ejemplo), una planificación clásica u orientada a recursos tanto homogéneos (clúster) como

heterogéneos (Grid) puede no ser efectiva al no estar contemplada esta característica tan particular de esos sistemas.

El proyecto *Korea@home* sigue la filosofía del proyecto *Seti@home* en cuanto a la compartición de los ciclos de reloj desperdiciados de los equipos de sobremesa para realizar trabajos de supercomputación con carácter social: búsqueda de enfermedades biomédicas, previsión de catástrofes meteorológicas…

Community Grids

Este tipo de sistemas distribuidos en Grid es el resultado de la cooperación entre varias organizaciones formando Organizaciones Virtuales. Su principal objetivo es minimizar el *makespan* del *workflow* que se quiere procesar. El algoritmo de planificación en el que está basado es *best-effort* y siempre va a tratar de minimizar el tiempo de ejecución sin importar el coste que pueda acarrear.

Dentro de esta estrategia, los métodos de planificación pueden estar basados en heurísticas o en meta-heurísticas. La figura 1.3 muestra una clasificación de las distintas posibilidades.

Service-Oriented Grids

Con una estrategia totalmente opuesta a los *Community Grid*, este tipo de Grid pretenden maximizar la satisfacción de los clientes (representada mediante el indicador QoS, *Quality of Service*).

Cada una de las tareas del workflow no puede continuar mientras sus dependientes no lo hayan hecho. Las condiciones para que puedan comenzar su ejecución vienen determinadas entre acuerdos cliente-proveedor representados mediante SLA (*Service Level Agreement*).

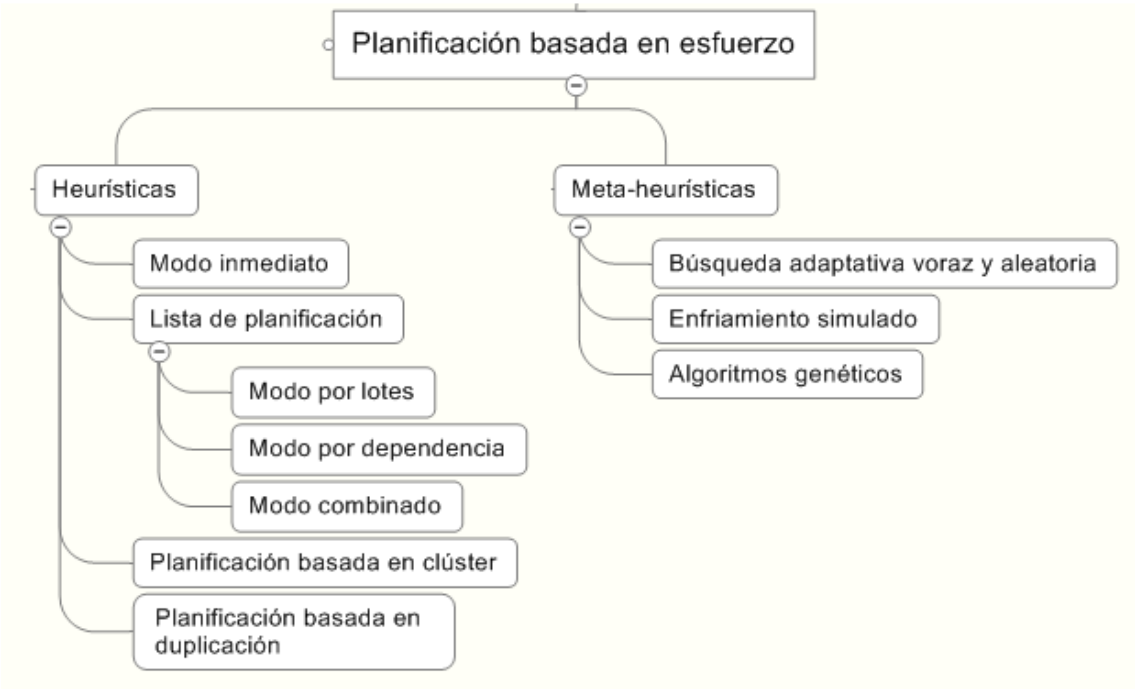


***Figura 1.3**. Planificación basada en esfuerzo para Community Grids. Adaptado de (Jia, Rajkumar, & Kotagiri, 2008).*

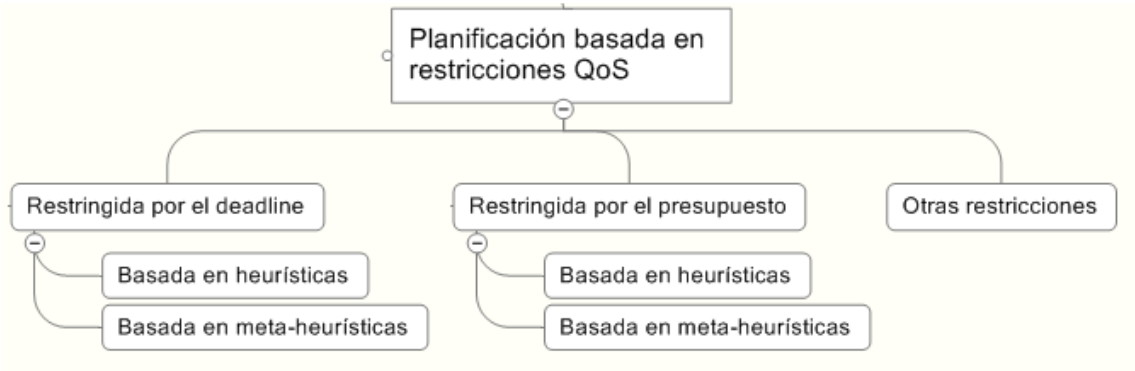


***Figura 1.4.** Planificación basada en esfuerzo para Service-Oriented Grids..*

Este tipo de Grid están orientados al modelo de *Utility computing* y, a diferencia de los *Community Grid*, su objetivo es obtener un beneficio económico dentro de las organizaciones.

En la figura 1.4, puede verse un esquema de las estrategias utilizadas dentro de la planificación basada en QoS.

4. Fundamentos de la planificación de sistemas distribuidos en Grid

Dentro de las técnicas de planificación y antes de llegar a hablar de la planificación en sistemas distribuidos en Grid, se tratará el tema de la planificación de supercomputadores.

No se debe confundir un supercomputador basados en clúster con un sistema distribuido en Grid. Los supercomputadores están compuestos por cientos de procesadores que comparten una misma memoria y que están interconectados por una red de alta velocidad (LaTorre, Peña, Robles, & de Miguel, 2008). Las tareas que se quieren ejecutar en este tipo de sistemas suelen ser altamente paralelizables de forma que el problema de la planificación de tareas en un supercomputador consiste en encontrar la mejor secuencia de ejecución de las tareas en los distintos procesadores minimizando el tiempo de ejecución y teniendo en cuenta las restricciones y dependencias entre las tareas y los procesadores.

Ya vimos en su capítulo correspondiente que el problema de la planificación trata de ocupar una serie de recursos para la realización de un conjunto de tareas en un determinado tiempo. El problema viene caracterizado por tres componentes:

- El número de recursos y el número de tareas que se quieren realizar.
- El conjunto de restricciones que se deben satisfacer.
- La función objetivo que se debe optimizar.

En cuanto a la optimización de esta planificación, en (Iordache, Boboila, Pop, & Stratan, 2008) se indica que en función de las características del sistema distribuido en Grid, la optimización puede orientarse hacia:

- Maximizar el rendimiento del sistema y la satisfacción del usuario (*QoS*).
- Maximizar la autonomía de cada uno de los recursos.
- Maximizar la escalabilidad del sistema.

- Maximizar la tolerancia a fallos del sistema.

Por otra parte, los problemas de planificación, pueden estar orientados a optimizar:

- Un único objetivo
- Varios objetivos: En (Shimizu & Tanaka), se plantea de manera formal el problema de optimización siguiendo varios objetivos.

La planificación de sistemas distribuidos en Grid, ineludiblemente ha surgido como evolución de sistemas homogéneos basados en clúster. Debido a sus diferencias, los retos a resolver, según (Jia, Rajkumar, & Kotagiri, 2008), a este respecto son:

- Los recursos del Grid están compartidos y los usuarios compiten por ellos.
- Los recursos no los controla el planificador.
- Los recursos son heterogéneos y no se comportan de igual manera a la hora de trabajar con una determinada tarea.
- Las tareas intensas en datos tienen que transmitir grandes cantidades de datos.

La pregunta que trataremos de resolver es:

> "*How to make use of millions of computers world-wide, ranging from simple laptops, to clusters of computers and supercomputers connected through heterogenous networks in an efficient, secure and reliable manner?" (Xhafa & Abraham, 2008)*

Según (Xhafa & Abraham, 2008), la planificación Grid puede verse como un conjunto de problemas debido a la enorme cantidad de parámetros que intervienen en la misma así como la distinta naturaleza de las aplicaciones que pueden funcionar en entornos Grid. El problema de encontrar un valor óptimo en un problema de planificación sobre recursos heterogéneos, en general, es de tipo NP-duro.

Dentro de las posibles planificaciones, existe un aspecto importante y es si las tareas que se van a planificar son independientes o tienen dependencias entre ellas. En el primer caso, el escenario típico está más cercano a los supercomputadores. Aquí, las tareas no tienen dependencias y pueden ejecutarse concurrentemente o de forma paralela en los distintos recursos. Cuando los trabajos son la composición de distintas tareas y existe interdependencia entre ellas, solemos hablar de workflow y el problema es diferente. En este segundo caso, como se indica en (Jia, Rajkumar, & Kotagiri, 2008), nos vemos frente a un problema que no puede resolverse de forma óptima por debajo de un tiempo polinomial. A estos problemas se les conoce como NP-completos.

El problema de planificación de tareas en un sistema distribuido en Grid es un problema intratable, lo que hace la necesidad de utilizar heurísticas o meta-heurísticas para obtener resultados sub-óptimos (Iordache, Boboila, Pop, & Stratan, 2008).

Dentro del problema de planificación Grid, según (Xhafa & Abraham, Meta-heuristics for Grid Scheduling Problems, 2008) podemos tratar de optimizar indicadores de forma independiente o tratar de optimizar varios objetivos.

En las dos siguientes secciones se presentarán indicadores apropiados para la optimización basada en un único objetivo, así como en varios objetivos con sus ecuaciones correspondientes que nos servirán para hacernos una idea de las distintas posibilidades ya existentes y decidir en base a cual plantearemos nuestra propuesta de planificación SQS.

PLANIFICACIÓN BASADA EN UN ÚNICO OBJETIVO

Según (Xhafa & Abraham, Meta-heuristics for Grid Scheduling Problems, 2008), la idea tras el diseño de un planificador es optimizar alguna de las variables clave que afectan al comportamiento del mismo. En este artículo se hace clara distinción dentro de la planificación basada en un único objetivo en los puntos de vista de los dos actores principales que participan en una planificación: el punto de vista del cliente y el del proveedor de servicios.

Desde el punto de vista del primero, lo interesante es que el planificador devuelva resultados que minimicen el tiempo que tarda en finalizarse la última tarea que se ha enviado (*makespan*) o que se minimice el tiempo suma de finalizar todas y cada una de las tareas (*flowtime*)

A continuación, pueden verse las dos ecuaciones que modelan esta minimización:

$$Minimización\ de\ makespan: \min_{S_i\ \epsilon\ Plan} \left\{ \max_{j\epsilon\ Trabajos} F_j \right\}$$

Ecuación 1.1. *Ecuación para la minimización de makespan.*

$$Minimización\ de\ flowtime: \min_{S_i\ \epsilon\ Plan} \left\{ \sum_{j\epsilon\ Trabajos} F_j \right\}$$

Ecuación 1.2. *Ecuación para la minimización de flowtime.*

Donde *Fj* es el tiempo en finalizar la tarea *j*, Plan es el conjunto de posibles planificaciones y Trabajos es el conjunto de trabajos que hay que planificar.

Otra de las métricas relacionadas con el tiempo en finalizar tareas es el tiempo en completar una tarea. Esta métrica se define como el tiempo que es necesario para completar todas las tareas que un recurso tiene asignado, así como la tarea que se acaba de asignar. Esta métrica cuantifica el rendimiento del recurso considerando su historia pasada.

La modelización de esta métrica depende del modelo de planificación que se esté utilizando. Además, esta métrica es una de las más apropiadas para modelos de planificación basados en aproximación como será el objetivo de este trabajo.

Desde la perspectiva del proveedor de recursos, la métrica más importante a optimizar es el uso de los mismos. Una variable utilizada normalmente para medir el uso

de cada recurso suele ser el uso medio de los recursos que puede definirse de la siguiente manera:

$$Uso\ medio\ del\ recurso = \frac{\{\sum_{\{i\ \epsilon\ Recursos\}} FinTarea\ (i)}{makespan\ x\ \#recursos}$$

Ecuación 1.3. *Ecuación para el cálculo del uso medio de un recurso.*

donde *FinTarea(i)* es el tiempo que tarda el recurso *i* en finalizar una determinada tarea, *i* es cada uno de los recursos del Grid, *makespan* es el tiempo en finalizar la última tarea y *#recursos* es la cantidad de recursos del Grid.

De igual manera que el tiempo en completar una tarea era apropiado para los modelos de planificación que contaban con el historial de ejecución del recurso, será interesante para un sistema de planificación aproximado no contar únicamente con las características de recursos y tareas de forma independiente, sino tratar de ajustar al máximo las características necesarias de la tarea al tipo de recurso donde se lanzará a ejecución.

Si introducimos la variable de prioridad en las anteriores métricas, estaremos permitiendo que una tarea que haya llegado más tarde a ser planificada y que tenga mayor prioridad, se ejecute antes o en un recurso mejor. Esta variable de prioridad es clave en los sistemas Grid desde el punto de vista del proveedor ya que a él le interesa optimizar el beneficio económico y dar mayor prioridad a un cliente importante que a uno de quien obtiene menor beneficio puede ayudar a maximizar este beneficio. Un ejemplo lo obtenemos de la modificación del tiempo en completar las tareas por parte de un determinado recurso:

$$Tiempo\ en\ completar\ una\ tarea\ con\ prioridad = \sum_{j\ \epsilon\ Trabajos} w_j F_j$$

Ecuación 1.4. *Ecuación para el cálculo del tiempo en completar una tarea con prioridad.*

donde *wj* es la prioridad del trabajo *j* y *Fj* es el tiempo en terminar la tarea.

PLANIFICACIÓN BASADA EN VARIOS OBJETIVOS

Aunque en la sección anterior se haya tratado la optimización de un único parámetro, la naturaleza del problema de planificación suele llevarnos a tener que optimizar varios objetivos y, sobre todo, desde puntos de vista diferentes e incluso opuestos como puede ser el optimizar los objetivos de un cliente y de un proveedor.

Normalmente, la planificación que requieren optimización de varios objetivos, suelen dividirse en: (a) basadas en jerarquía, donde se priorizan los objetivos que se quieren optimizar y se van teniendo en cuenta según ese orden y (b) planificación simultánea, donde se tienen todos los objetivos en cuenta por igual y se trata de optimizar con igual prioridad

PLANIFICACIÓN ESTÁTICA VS PLANIFICACIÓN DINÁMICA

Dentro de las estrategias de planificación, una de las primeras decisiones que hemos de tomar es si es suficiente una planificación estática o los requisitos del sistema imponen que sea dinámica.

En informes como (Wu, Yu, Jin, Lin, & Schiavone, 2004), (Theys, Braun, Siegal, Maciejewski, & Kwok, 2001) (Ramamritham, 1993) se han elegido estrategias de planificación estática en la que la asignación de tareas se realiza a priori y donde no se consideran la posibilidad de fallo, la carga en exceso de un nodo o la inanición de otro.

Las técnicas de planificación dinámica (Mahmood, 2000), (Page & Naughton, 2005), (Prodan & Fahringer, 2005) (Greene, 2001) consiste en asignar las tareas dinámicamente a su llegada. Esta estrategia se utiliza cuando es difícil estimar el coste de las aplicaciones o cuando los trabajos llegan sobre la marcha (*online scheduling*). La planificación de tareas dinámica tiene dos componentes principales:

Estimación del estado del sistema a diferencia de la estimación de coste de la planificación estática. Esto implica recolectar información sobre los recursos del Grid.

La toma de decisiones.

Uno de los puntos clave de las técnicas de planificación dinámica es elegir la estrategia para la estimación de los recursos del Grid y la posterior toma de decisiones. En cuanto a la primera, podemos elegir entre las estrategias centralizada y descentralizada:

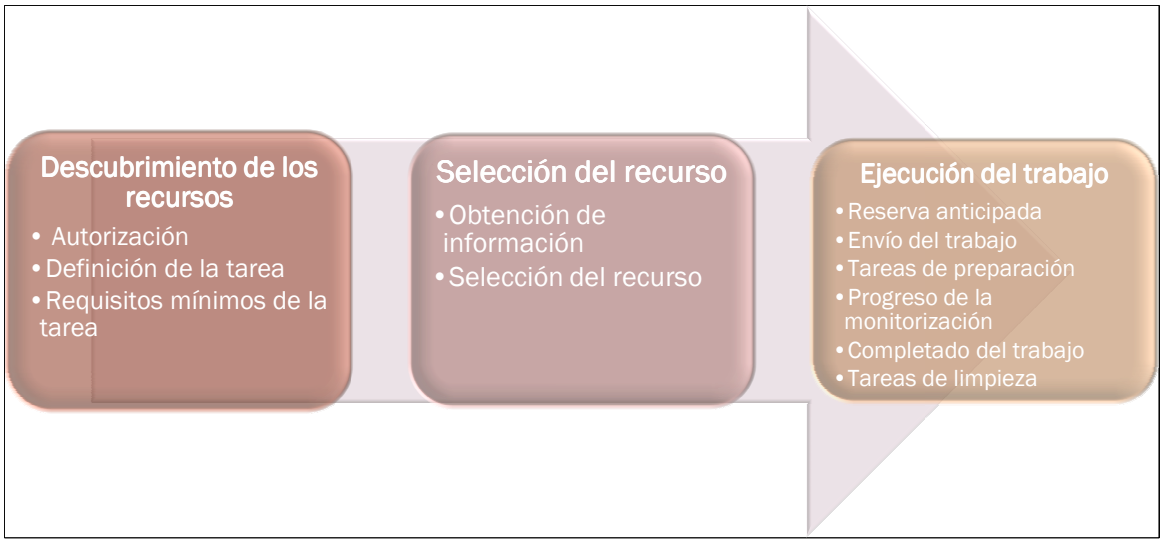
- **Centralizada** (Wu, Yu, Jin, Lin, & Schiavone, 2004), (Mahmood, 2000), (Page & Naughton, 2005), (Zomaya, Ward, & Macey) en el que un recurso recoge la información de carga del sistema y calcula la planificación óptima. Las ventajas de esta estrategia son la rapidez la sencillez de manejo, el despliegue sencillo y la posibilidad de reubicar recursos. Sin embargo, debido a la naturaleza descentralizada de los entornos Grid, la estrategia centralizada no proporciona escalabilidad, robustez ni tolerancia a fallos. El caso del meta-planificador del proyecto *GridWay*
- **Descentralizada** (Csaji, Monostori, & Kadar, 2004), (Seredynski, Koronacki, & Janikow, 1999), (Weichhart, Affenzeller, Reitbauer, & Wagner, 2004), (Cao, Spooner, Jarvis, Saini, & Nudd, 2004) con el problema añadido de los elevados costes en comunicaciones de red. Lo que se hace es particionar el sistema en diferentes Grid más pequeños con su planificador local correspondiente y que intercambian información para la gestión global del mismo. Esta estrategia también permite que cada planificador local disponga de sus propias políticas lo que se alinea muy bien con la idea de las Organizaciones Virtuales. Como ejemplo del uso de esta estrategia, se dispone del proyecto DIOGENES (Iordache, Boboila, Pop, & Stratan, 2008) donde se tiene en cuenta los dos problemas principales de un planificador Grid descentralizado, a saber, (a) cómo monitorizar los recursos y (b) cómo descubrir los servicios disponibles.

5. Etapas dentro del problema de la planificación

Según (Schopf, 2006), existen diez fases en las que se van a realizar la planificación de una tarea en un sistema distribuido en Grid. Estas diez fases se agrupan dentro de tres clases principales, a saber, (a) descubrimiento de recursos, (b) selección del

recurso más apropiado y (c) ejecución de la tarea sobre ese recurso. La figura 1.5 muestra un resumen de estas tareas, junto con cada una de las fases que hay en su interior.

Las tres fases que pueden verse en la figura 1.5 comienzan con una vez se ha recibido la solicitud de ejecución de una determinada tarea al planificador. Lo primero que ha de saber es de los recursos que se dispone para elegir el mejor en la segunda fase y lanzarlo a ejecución en la tercera.



do en Grid (fuente

Dentro de la fase de descubrimiento de los recursos disponibles, se han de realizar tareas con las que conseguir acceder de forma remota a los recursos (fase de autorización) para conseguir así la información relevante sobre el estado de cada recurso. Posteriormente se ha de analizar las necesidades y características de la tarea para, en función de esta, elegir el recurso más apropiado para que se cumplan las restricciones temporales o de cualquier otra índole impuestas a priori a la misma.

En la fase de selección, una vez se dispone de acceso a cada uno de los recursos, se solicita información sobre su estado y disponibilidad. Con esta información junto a la información sobre la tarea, se puede realizar la elección del recurso. Este es el elemento

clave de la planificación y es aquí donde las distintas estrategias de planificación han de demostrar su potencia y eficiencia.

Una vez elegido el recurso, se envía a ejecución al recurso. En esta fase, se ha de haber realizado una reserva anticipada, para que ningún otro planificador o proceso local pueda ocupar el espacio que el planificador entiende como para él, se han de realizar las tareas de preparación del entorno, transmisión de datos y finalmente ejecución. Durante esta ejecución se monitorizan los datos que van a servir para guiar el buen procesamiento de la tarea según la hipótesis de selección. Además, una vez se haya terminado de ejecutar la tarea, se obtendrán los resultados reales de la ejecución que confirmarán si la hipótesis de selección fue correcta o no. También hay que tener en cuenta la limpieza del sistema tras la ejecución para que quede todo listo para futuras tareas a ejecutar.

6. La Inteligencia Artificial aplicada al problema de planificación

Ya se ha revisado la teoría y técnicas de planificación y, en concreto en sistemas distribuidos en Grid. También se han presentado sus fundamentos así como sus características y problemática. Ahora, vamos a centrarnos en el problema de la planificación de tareas en sistemas distribuidos en Grid.

Antes de dejar que SQS realice su propuesta de planificación para tales sistemas, se va a presentar en esta sección un análisis detallado de las distintas estrategias encontradas hasta la fecha en la literatura que servirá de punto de partida para la justificación de la elección realizada para nuestro sistema planificador.

Nuestro interés, como se justificará más adelante, se centra en la planificación dinámica. Ya se ha visto que este es un problema de complejidad NP-duro o NP-completo. Por ello, la necesidad de heurísticas o meta-heurísticas se vuelve imprescindible a la hora de encontrar un método con el que decidir qué recurso ejecutar qué tarea.

Es por todo esto, que el uso de la Inteligencia Artificial y más concretamente en las técnicas de *soft-computing* en las que nos vamos a centrar, toman mayor relevancia. A

lo largo de toda esta sección se revisará la literatura que más hincapié ha hecho en esta línea.

PLANIFICACIÓN UTILIZANDO LÓGICA DIFUSA

Fundamentos de lógica difusa

El concepto de "lógica difusa" fue presentado por primera vez en el artículo *Fuzzy Sets* (Zadeh, 1965). Esta teoría generaliza la lógica binaria clásica (bivalente) en la que los fenómenos pueden tomar únicamente valores 0 o 1. Aunque se han propuesto lógicas con tres valores (trivalentes) o muchos más (multivalentes), la lógica difusa se centra en trabajar con conjuntos borrosos. Un conjunto borroso sobre un universo, asigna un grado de pertenencia en el intervalo [0, 1] a cada elemento del universo.

Esta lógica se separa radicalmente de la modelización que trata de hacer la matemática de la realidad y pretende modelizar más fehacientemente el mundo en el que vivimos. De tal manera, los conjuntos borrosos modelan conceptos como "meses fríos", "meses calurosos", "hombres altos", "temperatura muy bajas"… sin tener que trabajar con información completamente nítida como que un mes sólo puede ser frío (valor 0) o puede ser caluroso (valor 1).

Las operaciones de la lógica difusa, generalizan las operaciones clásicas: unión, intersección, diferencia, negación y complemento, así como ciertas otras operaciones particulares de los subconjuntos difusos.

Para cada variable difusa que se quiera utilizar, habrá que definir los conjuntos difusos sobre ese universo así como el grado de pertenencia de los mismos a cada variable difusa. Los grados de pertenencia vienen representados por funciones trapezoidales, lineales o curvas dependiendo del caso y la aplicación de estas funciones a las variables se conoce como *borrosificación*.

Una vez definidas los conjuntos difusos y las funciones de pertenencia, la manera que un sistema experto difuso tiene de razonar es mediante reglas difusas de la forma *if…then* donde tanto antecedentes como consecuentes son variables difusas y el resultado

de cada regla genera un área resultado del solapamiento de los áreas que representan en las funciones de pertenencia el grado de pertenencia de cada variable a cada conjunto.

Esta área resultada ha de convertirse en un valor nítido y ya no difuso que sirva para modelizar de nuevo la realidad. Los métodos más utilizados para obtener este valor (*desborrosificación*) son el centro de gravedad y el método max-min.

Las reglas que determinan los predicados tanto difusos como nítidos mediante procesos de *borrosificación* y *desborrosificación* se construyen con base en fuentes de datos basados en la historia pasada, el entrenamiento de redes neuronales difusas y aproximación numérica.

Lógica difusa aplicada al problema de planificación

La potencia de la lógica difusa viene dada por la posibilidad de obtener conclusiones y generar respuestas basadas en datos cualitativos a la par que imprecisos, ambiguos o incompletos.

La información que podemos obtener de una tarea que quiera ejecutarse en un sistema distribuido en Grid a priori tiene un alto nivel de imprecisión y ambigüedad. Ya se vio como en (Schopf, 2006) se hablaba del problema de la falta de información sobre las necesidades computacionales de cualquier tarea para que un planificador pudiese utilizar dicha información a la hora de encontrar el mejor recurso para ella.

En algunos trabajos como en (Wang, Yuan, & Liu, 2008) se han utilizado como variables difusas métricas relacionadas con la satisfacción del cliente, el tiempo de procesamiento o la fecha de entrega ya que son las variables más importantes dentro de los procesos de producción industrial y desde el punto de vista del cliente son las variables que se desean optimizar.

Desde el punto de vista del proveedor, también hay literatura en la que se ha utilizado lógica difusa. Este es el caso de (Yu, Luo, Chou, Chen, & Zhou, 2007) y de (Tarancón, Garmendia, & Santos, 2009) donde se ha utilizado lógica difusa para clasificar el estado de un determinado recurso en función de variables que cualquier sistema puede proporcionar, como el uso de CPU, la cantidad de memoria RAM utilizada o el ancho de

banda disponible. Optimizar estas variables resulta clave para el punto de vista del proveedor de servicios en cuanto a que sus recursos estén utilizados el máximo tiempo posible lo más que se pueda y que terminen las tareas lo más temprano posible para poder ejecutar nuevas tareas que aumenten su beneficio económico.

PLANIFICACIÓN UTILIZANDO REDES NEURONALES

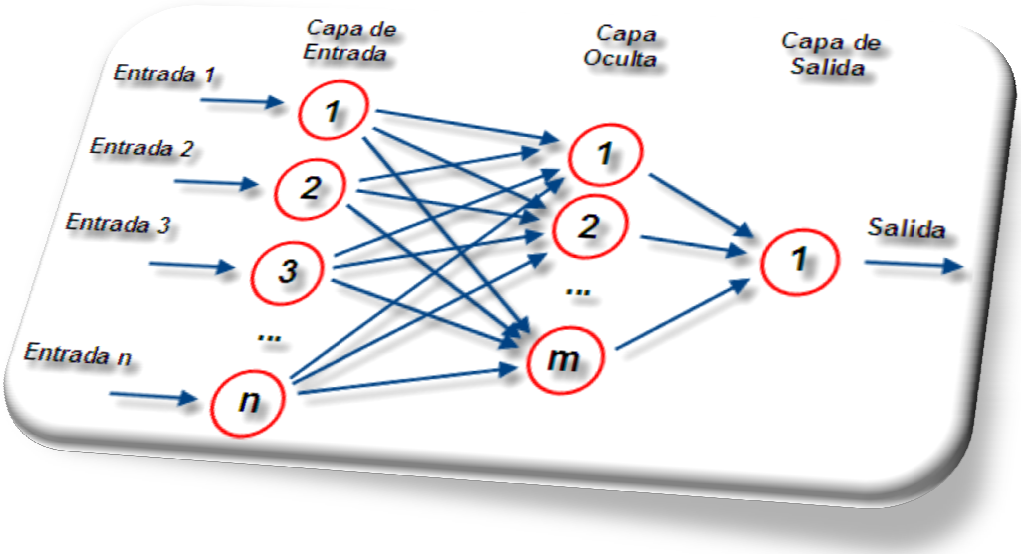
Fundamentos de redes neuronales

Las redes neuronales son un paradigma de aprendizaje automático inspirado en las interconexiones neuronales de los seres vivos. Se fundamentan en la naturaleza de los cerebros biológicos en donde la información no reside en una determinada unidad de almacenamiento. En realidad está repartida entre todas las conexiones sinápticas y se codifica en función de configuración de las mismas al conectar las neuronas. La estructura general de una red neuronal se compone de diferentes entradas por las que se van a producir estímulos de información y que van a excitar las neuronas de las capas ocultas en donde se codifica la información aprendida. En la capa de salida se va obtener el resultado de los estímulos a la entrada y dependiendo de la topología de la red neuronal y del proceso de aprendizaje utilizado, se va a realimentar con ese resultado obtenido de nuevo a las neuronas de la capa oculta para que aprendan de este nuevo caso.

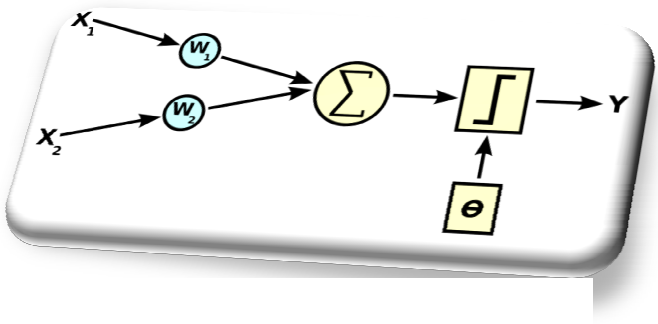
El aprendizaje en las redes neuronales se consigue, generalmente, por repetición de ejemplos. De igual manera que los seres vivos aprenden determinados comportamientos mediante unas condiciones de entrada y un resultado a la salida, las redes neuronales van a organizar las conexiones sinápticas de manera que, dadas unas entradas y unas salidas conocidas a priori, después de ciertas épocas de ejemplos, las conexiones sinápticas estén organizadas de tal manera que al introducir un patrón parecido en la entrada, se origine algún patrón parecido a alguna de las salidas.

Un claro ejemplo de uso de redes neuronales lo encontramos en el campo de la visión artificial por computador. Seguro que alguna vez hemos entrado en el parking de un centro comercial y al coger el ticket nos han hecho una fotografía de la matrícula que después ha sido impresa en el propio ticket. La forma en que el sistema es capaz de reconocer los dígitos y las letras de nuestra matrícula es gracias al entrenamiento de

conjuntos de imágenes que representan números y que expertos le indican al sistema si ese conjunto de píxeles se trata de un uno, un cero o la letra A.



La estructura de las redes neuronales suele dividirse en capas (de entrada, ocultas y de salida, generalmente) y cada capa está compuesta por neuronas que se interconectan con las neuronas de la capa anterior y de la siguiente.



Funciones de una neurona de tipo perceptrón.

De tal forma, cada neurona va a procesar las conexiones que le llegan y a transmitir un resultado a la siguiente capa mediante su conexión de salida. Para que este sistema funcione se han de definir las siguientes funciones:

- $\sum$ **Función de propagación o de excitación**, que suele ser una función de suma de todas las variables de entrada multiplicada por el peso (cada uno de los w_i de la figura 1.7) que cada una de las entradas tiene sobre la neurona. Cuando el resultado de esta operación es positivo, la neurona es **excitadora**, mientras que si es negativo, es **inhibidora**.

- θ **Una función de activación**, que modifica a la anterior y que puede o no existir, en cuyo caso, sería la misma que la de propagación.

- $\int$ **Una función de transferencia**, que se aplica al valor de salida y que depende del modelo que estamos utilizando utilizaremos funciones que den valores entre 0 y 1, como en el caso de la **función sigmoidea** o entre -1 y 1, como es el caso de la **función tangente hiperbólica**.

En lo que respecta al aprendizaje, hay dos tipos principales: el aprendizaje supervisado y el no supervisado. La diferencia radica en que en el aprendizaje supervisado se conocen a priori tanto entradas como salidas y sólo hay que iterar el conjunto de aprendizaje a la red neuronal indicándole la respuesta para que aprenda. Sin embargo, en el aprendizaje no supervisado, los datos de salida no se conocen a priori y se van refinando a medida que se van conociendo nuevas soluciones y se comparan con las características que tienen las soluciones que se consideran óptimas o, al menos, aceptables. Este último tipo de aprendizaje no suele dar la mejor solución, pero sí suele dar la solución más innovadora.

Redes neuronales aplicadas al problema de planificación

Dentro de la literatura revisada, se han encontrado casos de uso de redes neuronales para diseñar planificadores de sistemas distribuidos en Grid. En concreto, en (Yu, Luo, Chou, Chen, & Zhou, 2007), se han empleado redes neuronales difusas dentro del problema de planificación para la adaptación automática de las funciones miembro.

Las redes neuronales difusas combinan el poder trabajar con incertidumbre de la lógica difusa con las capacidades de aprendizaje que ofrecen las reglas neuronales. La cuestión en este caso es codificar cada una de las reglas del estilo *if…then* difusas dentro de las neuronas de la capa oculta y que sus pesos utilicen valores difusos que se *borrosifican* y se *desborrosifican* en las capas de entrada y de salida.

En cuanto a las estrategias a la hora de construir una red neuronal, encontramos, por ejemplo en (Yu, Luo, Chou, Chen, & Zhou, 2007) que se han utilizado cinco capas: de entrada, tres ocultas y de salida con estrategia *feed-forward* y *back-propagation* de forma que cada cinco veces que se produce la primera en cada nodo, se saca la media de cada nodo, de todo el sistema y se aplica a la ecuación.

PLANIFICACIÓN UTILIZANDO ALGORITMOS GENÉTICOS

Los trabajos de Darwin por demostrar el porqué de la evolución de los seres vivos, le llevó en 1959 a publicar su conocida obra sobre la selección natural (Darwin, 1959). Esta teoría, inicialmente estuvo limitada al campo de la biología y a los seres vivos.

Con el desarrollo de los sistemas informáticos, se empezaron a aplicar estas teorías en algoritmos. Al igual que en el caso de los seres vivos, se les hace evolucionar para que, partiendo de un problema, vaya evolucionando la representación del mismo hasta llegar a encontrar una solución suficientemente buena.

Si obtener una planificación de trabajos en recursos ya se ha comentado que tiene una complejidad muy elevada, los algoritmos genéticos pueden ser buenos aliados a la hora de, a partir de una asignación no óptima de trabajos en recursos, hacerla evolucionar hasta encontrar esa deseada solución óptima.

En esta sección, se va a presentar esta tercera rama del *soft-computing* y en particular de su aplicación al problema de planificación, objeto de este estudio.

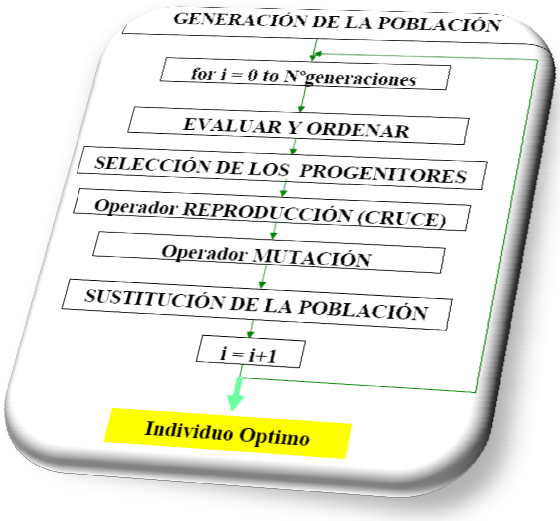
Fundamentos de Algoritmos Genéticos

El concepto de algoritmo genético (Bagley, 1967) no es nuevo y ha tenido diferentes momentos a lo largo de la historia en el que su estudio ha fascinado a

bioinformáticos de todo el mundo. El padre de los algoritmos genéticos (Holland, 1962) fue quien habló por primera vez de este concepto.

> *Un conjunto inicial de individuos que evoluciona a lo largo de un número de generaciones mediante técnicas de reproducción y mutaciones, para aproximarse al individuo óptimo, el que maximiza o minimiza una función de comportamiento o ajuste. (Goldberg, 1898)*

La esencia de estos algoritmos es hacer evolucionar una población inicial modificando algunos de sus propiedades hasta que consigan una solución al problema lo suficientemente buena y aparezca el individuo solución.



)

Los algoritmos genéticos son algoritmos de búsqueda basados en probabilidad. Su fundamento es que, si se itera las veces suficientes las mutaciones y selecciones se convergirán a la solución óptima.En general, las fases por las que tiene que pasar un algoritmo genético son las representadas en la figura 1.7.

En la figura 1.8, puede verse claramente las distintas fases de un algoritmo genético que a continuación detallamos:

- **Fase de inicialización de la población**: Esta fase parte de una buena codificación del problema en los cromosomas con los que se va a trabajar. Lo ideal es codificarlos en valores sencillos como es el caso de los valores binarios. Para la inicialización, basta con una selección aleatoria, pero hay que tener en cuenta que la población inicial no caiga fuera del conjunto de resultados posibles.
- **Fase de evaluación del objetivo**: A cada individuo de la población se le evalúa para saber sus "potencial" supervivencia y se clasifican todos ellos siguiendo esa función de evaluación.
- **Fase de reproducción**: Una vez se dispone de la clasificación de los candidatos, se han de:
    - **Elegir los padres** de la siguiente generación. Esta selección también depende del problema que se trate de resolver.
    - **Reproducción de estos padres** originando con la función de reproducción unos nuevos hijos.
    - **Mutación** de esos nuevos hijos de forma aleatoria.
- **Comprobación de si la nueva población mejora a la anterior o la empeora**. En el primer caso, se vuele a ejecutar el algoritmo sobre esta nueva población, mientras que en el segundo caso, se sigue con el caso inicial y se desecha la nueva.

Algoritmos Genéticos aplicados al problema de planificación

Los algoritmos genéticos son, sin duda, la rama *soft-computing* donde más investigación se ha realizado para el problema de planificación y, más concretamente, a la resolución de planificaciones de tareas en sistemas distribuidos en Grid.

Generalmente se va a tratar de encontrar un orden óptimo de ejecución de las tareas sobre los recursos disponibles haciendo evolucionar una posible planificación hasta encontrar la combinación que mejor se aproxime al valor óptimo.

Hay casos en la literatura (LaTorre, Peña, Robles, & de Miguel, 2008) donde se ha planteado la resolución de problemas desde este punto de vista. Este es el caso del problema SCS (*SuperComputing Scheduling*) visto como un problema combinatorio donde el objetivo es encontrar el orden de la secuencia de trabajos más apropiado manteniendo el cumplimiento de las restricciones dadas. En este caso, el objetivo es maximizar el uso de CPU.

2. SQS - SUPERCOMPUTING QUALITY SCHEDULER

Ya se ha presentado la controversia que genera el problema de planificación. Se han revisado las distintas perspectivas y los distintos puntos de vista a la hora de plantear el problema. También se ha visto el caso particular de planificación en sistemas distribuidos en Grid y sus características específicas. Así mismo, se ha corroborado cómo los algoritmos genéticos son la principal estrategia *soft-computing* a la hora de optimizar de forma inteligente el problema de la planificación de sistemas distribuidos en Grid.

Llega el momento de presentar el *framework* SQS. Se trata de un *framework* para la clasificación y posterior planificación de tareas en recursos organizados en Grid. Su filosofía plantea una alternativa totalmente novedosa e innovadora para el problema de planificación aplicado a este tipo de sistemas distribuidos. Su propuesta, lejos de tratar el problema de planificación como un problema de optimización multi-objetivo (como en los algoritmos genéticos), lo va a tratar como un problema de clasificación utilizando lógica difusa. Es más, no sólo se va a conformar con utilizar un sistema experto difuso

que decida a qué recurso enviar cada tarea. Se utilizará la potencia de las redes neuronales para afinar la clasificación aprendiendo de los resultados de ejecución y comparándolos con las hipótesis formuladas. De esa manera, conseguiremos aprovecharnos de datos ocultos que son difíciles de considerar al estar intrínsecamente vinculados a la ejecución particular de cada tarea sobre unos y otros recursos. A medida que se planifiquen más y más trabajos, el sistema aprenderá haciendo que la clasificación y, por tanto, la planificación, sea cada vez más óptima.

En su propuesta para diseñar algoritmos inteligentes para la planificación de sistemas en Grid (Huang, French, Maple, & Bessis, 2006), se proponían cuatro fases para el desarrollo de su propuesta. En nuestro caso, nos apoyaremos en dos de ellas: La primera analizará los factores más importantes a la hora de planificar trabajos en un sistema distribuido en Grid. La segunda, desarrollará los algoritmos inteligentes.

En primer lugar se van a presentar los resultados del análisis de los factores clave de la planificación de sistemas distribuidos en Grid. Después, se presentará la arquitectura propuesta para SQS haciendo especial hincapié en los sistemas neuro-difusos que son fundamentales de esta estrategia de planificación planteada. En tercer lugar, se presentará el método elegido para la toma de decisiones basado en conjuntos borrosos. Finalmente, se presentará la estrategia de aprendizaje que permitirá que los resultados de selección sean cada vez más precisos y óptimos a lo largo del tiempo.

1. Análisis de factores clave para la planificación

Para poder planificar se necesitan datos. Si el problema que pretende resolver SQS se centra en la planificación de tareas en recursos, se ha de definir qué variables se van a considerar tanto para el caso de las tareas, como para el caso de los recursos.

MÉTRICAS PARA LA CLASIFICACIÓN DE TAREAS

Al buscar métricas para clasificar las tareas, nos encontramos con que la mayoría de planificadores de sistemas en Grid utilizan el lenguaje de definición de envío de trabajos JSDL (Brisard, Drescher, Ly, McGough, Pulsipher, & Savva, 2005). Por una parte, basarse en este formato garantiza un alto índice de integración con la mayoría de

middleware dedicados a la planificación de entornos en Grid ya que es el que la mayoría de sus planificadores utiliza. El caso de *GridWay* (Huedo, Montero, & Llorente, 2005) es un claro ejemplo de ello. Sin embargo, al analizar los campos disponibles en este lenguaje (véase la Figura 2.1), nos encontramos con que existen muy pocos para la definición del propio trabajo y que están más relacionados con la descripción de los recursos necesarios.

```
<Resources>
   <CandidateHosts ... />?
   <FileSystem .../>*
   <ExlusiveExecution .../>?
   <OperatingSystem .../>?
   <CPUArchitecture .../>?
   <IndividualCPUSpeed .../>?
   <IndividualCPUTime .../>?
   <IndividualCPUCount .../>?
   <IndividualNetworkBandwidth .../>?
   <IndividualPhysicalMemory .../>?
   <IndividualVirtualMemory .../>?
   <IndividualDiskSpace .../>?
   <TotalCPUTime .../>?
   <TotalCPUCount .../>?
   <TotalPhysicalMemory .../>?
   <TotalVirtualMemory .../>?
   <TotalDiskSpace .../>?
   <TotalResourceCount .../>?
   <xsd:any##other>*
</Resources>?
```

Figura 2.1. *Extracto relativo a la definición de recursos necesarios por un trabajo en lenguaje JDSL.*

Por otra parte, para definir recursos no es obligatorio utilizar el lenguaje JSDL. En realidad, cualquier definición de intercambio de información puede ser válida para que los planificadores puedan conocer el estado de determinado recurso. Será suficiente con utilizar un lenguaje estándar, como XML para ello. Como ejemplo, el que nos ofrece (Iordache, Boboila, Pop, & Stratan, 2008) para la definición del recurso (véase la figura 2.2).

Este lenguaje, de una forma muy parecida a JSDL, dispone de una primera parte donde se definen los parámetros particulares de la tarea, como puede se su identificador, su ruta, sus argumentos de entrada y de salida… Después, en la sección `<requirements/>` es donde se definen las necesidades en cuanto a memoria, potencia necesaria de la CPU, tiempo de procesamiento, fecha límite de ejecución…

```
<task>
  <taskId>24</taskId>
  <path>/home/student/pi/mpi999999999.sh</path>
  <arrivingDate>2007/05/04</arrivingDate>
  <arrivingTime>01:45:05</arrivingTime>
  <arguments>999999999</arguments>
  <input></input>
  <output>mpi.out</output>
  <error>mpi.err</error>
  <requirements>
    <memory>2.95MB</memory>
    <swapSpace>2.95MB</swapSpace>
    <cpuPower>2682.41MHZ</cpuPower>
    <processingTime>39</processingTime>
    <deadlineTime>2006/06/09 00:00:01</deadlineTime>
    <schedulePriority>10</schedulePriority>
  </requirements>
  <nrexec>1</nrexec>
</task>
```

Figura 2.2. *Ejemplo de definición de tarea y de requisitos.*

En definitiva, nuestro objetivo es conseguir un conjunto de métricas que nos permitan caracterizar las tareas en función de ellas. En la tabla 2.1 se puede ver una propuesta de indicadores que pueden ayudarnos a conseguir este objetivo. A colación de esto último, cabe destacar la extensibilidad del *framework* SQS a la hora de incorporar futuras métricas.

Tabla 2.1. *Definición de recursos dentro de la de una tarea.*

Nombre de la métrica	Descripción
Tamaño	Espacio (en KB) que ocupa el trabajo en el disco duro
Tipo del archivo	Extensión del trabajo y que permite saber si se trata de un archivo de procesamiento matemático, de renderizado de imágenes o de aplicación Web
Propietario	Nombre del propietario del trabajo
# de bucles	En el caso de trabajos estáticos y que se conoce el número de bucles a priori, medida de iteraciones en media que se realizan

# llamadas recursivas	En el caso de trabajos estáticos y que se conoce el número de llamadas recursivas a priori, medida de llamadas de este tipo, en media, que se realizan
Sistema operativo	El tipo de sistema operativo que requiere el trabajo
Arquitectura	La arquitectura *hardware* que necesita el trabajo
Permisos	Los permisos de lectura, escritura y ejecución que tiene una determinada tarea, trabajo o aplicación.

MÉTRICAS PARA LA CLASIFICACIÓN DE RECURSOS

Obtener información de utilidad que nos ayude a conocer el estado en el que se encuentra un recurso es, a priori, un tema mucho menos complicado que en el caso de las tareas que vimos en el apartado anterior. Cualquier monitor de sistemas proporciona información básica sobre el estado de recurso particular en tiempo real (véase la figura 2.3). Esta información puede aumentarse y completarse con multitud de métricas que ayudarán a definir de forma más explícita el estado de un recurso. En este último punto, vuelve a verse la extensibilidad del *framework* SQS para incorporar nuevas métricas para futuros casos de uso.

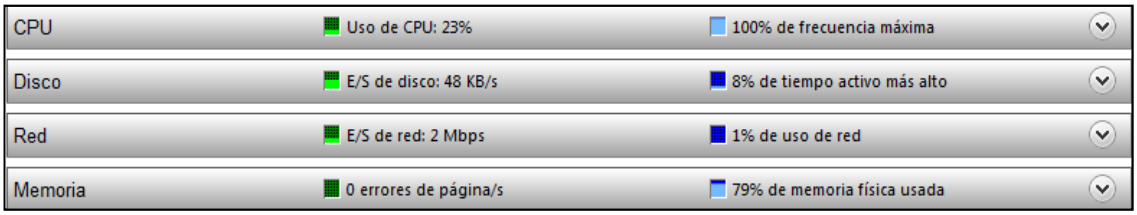


Figura 2.3. *Ejemplo del estado de un recurso utilizando el Monitor de Recursos de Windows.*

De forma paralela al lenguaje JSDL, existen diversas iniciativas para definir de una forma parecida los recursos de un sistema en Grid. Este es el caso de RDL (*Resource Description Language*). Este lenguaje no goza de la misma popularidad que JSDL y todavía se encuentra en una versión muy básica. No obstante, nos interesará esta perspectiva para componer nuestro formato de descripción de recursos. En la figura 2.4 se puede ver el esquema XML de la definición de un recurso en RDL.

```
<ResourceDefinition>
        <ClusterSoftware/>
        <AvailableSoftware/>?
        <Security/>?
        <CPU/>?
        <SystemMemory/>*
        <jsdl:FileSystem/>*
        <jsdl:OperatingSystem/>?
        <Network/>*
</ResourceDefinition>
```

Figura 2.4. *Definición de un recurso en lenguaje RDL.*

Cabe destacar de este ejemplo la forma en que se han definido los campos relativos a la CPU (`<CPU/>`), a la memoria RAM (`<SystemMemory/>`) y a la RED (`<Network/>`).

En la figura 2.5, puede verse un ejemplo de definición de una máquina *Apple* con dos procesadores y con arquitectura de 32 *bits*. En cuanto a las características de su procesador, el ejemplo nos indica que la velocidad de cada una de sus CPU es de 3 GHz, que el porcentaje de uso por parte del sistema es del 7,5 %, el uso del sistema para el usuario es del 12 % y que el porcentaje en que el recurso está ocioso es del 80,5 %.

```
<CPU>
        <jsdl:Architecture>x86_32</jsdl:Architecture>
        <Manufacturer>Apple</Manufacturer>
        <jsdl:IndividualCPUCount>2</jsdl:IndividualCPUCount>

        <Processor>
                <jsdl:IndividualCPUSpeed>3.0GHz</jsdl:IndividualCPUSpeed>
                <SystemPercent>7.5</SystemPercent>
                <UserPercent>12.0</UserPercent>
                <IdlePercent>80.50</IdlePercent>
        </Processor>
</CPU>
```

Figura 2.5. *Ejemplo de definición de la CPU de un recurso.*

Nótese en el hecho de que la definición tanto de la arquitectura (`<jsdl:Architecture/>`), el número de CPU (`<jsdl:IndividualCPUCount/>`) como la velocidad de cada procesador (`<jsdl:IndividualCPUSpeed/>`) coinciden con la manera

en que se definen en JSDL. Esto va a facilitar las comparativas y la asignación de tareas a recursos al compartir formato de definición.

De igual manera que para el caso de la CPU, se puede ver en la figura 2.6 cómo se define en lenguaje RDL la memoria RAM del sistema.

```
<SystemMemory>
        <Type>Physical</Type>
        <TotalBytes>1063174144</TotalBytes>
        <FreeBytes>395184128</FreeBytes>
</SystemMemory>
```

Figura 2.6. *Definición de la memoria RAM en lenguaje RDL.*

En este ejemplo, se indica que el sistema tiene memoria de tipo físico, la cantidad de memoria de que dispone (en *bytes*) y la cantidad de memoria que está disponible (también en *bytes*).

Por último, se muestra en la figura 2.7 el esquema de definición de las características de red de que dispone el recurso.

```
<Network>
        <jsdl:HostName/>*
        <jsdl:IndividualHostBandwidth/>?
        <TotalBytesSent/>?
        <TotalBytesReceived/>?
        <TotalBytesTransferred/>?
        <TotalActiveTransfers/>?
</Network>
```

Figura 2.7. *Definición de las características de red de un recurso.*

En este caso, tanto el nombre del recurso (`<jsdl:HostName/>`), como el ancho de banda del propio recurso (`<jsdl:IndividualHostBandwidth/>`) son informaciones compartidas con el lenguaje JSDL. A parte de ello, se puede obtener información sobre el total de información enviada, recibida y transferida, así como del total de transferencias activas de que dispone el recurso.

Es momento de proponer métricas para que SQS clasifique a los recursos en alguno de los tres grupos anteriormente mencionados. El análisis de estas métricas nos lleva a proponer como iniciales las que se resumen en la tabla 2.2. Al lado de cada una puede verse una pequeña descripción de su significado.

Tabla 2.2. *Métricas de clasificación de los recursos.*

Métrica	Descripción
% uso CPU	Porcentaje instantáneo de uso de CPU del recurso en el momento de la consulta
Velocidad CPU (Mhz)	Velocidad del procesador del recurso en MHz
Número de CPUs	Cantidad de procesadores de que dispone el recurso
RAM total	Cantidad total de memoria RAM que posee el recurso
% uso de RAM	Porcentaje instantáneo de uso de memoria RAM del recurso en el momento de la consulta
Ancho de banda	Capacidad total del ancho de banda del recurso
% de uso de ancho de banda	Porcentaje instantáneo de uso del ancho de banda del recurso en el momento de la consulta

2. Planteamiento del problema

Del análisis de las métricas de tareas, parece claro que, teniendo tantas, asignar una tarea a un determinado recurso que las cumpla parece un trabajo sencillo. No obstante, aparecen aquí dos problemas frecuentes en los planificadores de sistemas en Grid:

1. Es probable que no haya ningún recurso en el sistema distribuido en Grid que consiga satisfacer todas las necesidades definidas en el JSDL. Para este problema, podemos pensar en una solución sencilla: basta con relajar las

condiciones y las restricciones lo suficiente para conseguir que algún recurso sea suficientemente bueno como para que pueda ejecutar la tarea.

2. Digamos que cualquiera de los actores que propusimos en los escenarios de la introducción desea enviar un trabajo a un sistema en Grid. Rellenar un documento JSDL puede resultar un trabajo muy tedioso e incluso imposible de realizar. ¡Ninguno tiene porqué saber qué necesidades de recursos ha de tener su tarea! Es más, incluso sabiendo cómo rellenar el documento JSDL o teniendo información precisa de las necesidades de sus trabajos, no pasarían de ser meras estimaciones aproximadas que no tienen porqué ser ciertas ni si quiera acercarse. Como es evidente, este problema es más complejo de resolver y es donde se han centrado todos nuestros esfuerzos.

En el caso de los recursos, clasificarlos en función de las métricas que se definieron anteriormente parece tarea sencilla. Simplemente se ha de consultar esta información en el momento de diseñar la planificación y clasificarlos utilizando dichos valores.

Utilizar únicamente la información del estado de ese preciso instante hace que nos dejemos mucha información en el tintero acerca del comportamiento del recurso. ¿Qué ocurre si hemos consultado un determinado recurso en el único momento del día en que está ocioso? ¿Y si resulta que tiene un problema de hardware que le hace tener un comportamiento anómalo? ¿Y si existe algún proceso desbocado independiente a la ejecución de las tareas que hace que los datos de disponibilidad no sean fiables? Todos estos problemas aparecen al no considerar la historia del recurso en cuanto a su estado de ejecución. Al mismo tiempo, estamos obviando información muy interesante con la que podemos establecer patrones de comportamiento del recurso en función de la hora en que se encuentre (imagínese un recurso que esté siempre ocioso por las noches).

Para dar solución a los problemas de todos estos escenarios, SQS propone trabajar con información imprecisa y aproximada para dar un nuevo punto de vista al problema de planificación. Así mismo, la propuesta de SQS es construir un sistema neuro-difuso que aprenda del historial de ejecución de las tareas en los recursos y clasifique a ambos de manera que futuras planificaciones sean mucho más eficientes. Al combinar el uso de la

información aproximada con el aprendizaje a partir del historial de ejecución, conseguiremos ir un paso por delante del resto de las estrategias de planificación basadas en datos nítidos.

3. Solución al problema de planificación con SQS

La propuesta de SQS consiste en aunar la potencia de la clasificación neuro-difusa de tareas y recursos, con un novedoso sistema de selección del mejor recurso donde una tarea puede ejecutarse. Además, una vez se hayan obtenido los resultados reales de la ejecución, se utilizarán para valorar si la elección que previó SQS satisfizo suficientemente a clientes y proveedores y aprenderá de sus éxitos o sus errores.

Por una parte, el *framework* SQS va a realizar una doble clasificación difusa: de tareas y de recursos. Esta clasificación devolverá, en ambos casos, el grado de pertenencia a cada conjunto difuso. Los conjuntos difusos que vamos a manejar, serán ***math*** (para tareas de tipo matemáticas), ***render*** (cuando la tarea consista en renderizar imágenes vectoriales) y ***webapp*** (cuando la tarea sea una aplicación Web). Cada uno de estos conjuntos tendrá unas características diferenciadas dependiendo de las necesidades de CPU, de RAM y de RED que tengan. Cada una de estas características vendrá definida por sus correspondientes variables lingüísticas. De esta manera, las tareas quedarán especializadas en base a sus características ocultas y los recursos en función de su comportamiento, su historial de ejecuciones, su estado en un momento temporal determinado…

Una vez que tengamos clasificadas las tareas y los recursos, la asignación de unas a otros será natural y mucho más sencilla. La propuesta de SQS va a consistir en hacer una novedosa comparativa en términos de conjuntos difusos. Se va a utilizar un nuevo operador con el que se asegura el máximo nivel de ajuste entre las características de las tareas obtenidas a partir de la clasificación y el estado en el que se encuentre cada recurso. De tal manera, SQS será capaz de proponer una hipótesis al respecto de cual piensa, es la mejor forma de planificar las tareas en los recursos de manera que se garantice la máxima calidad de servicio y satisfacción tanto para el cliente como para el proveedor de servicios.

Finalmente, SQS delegará en el *middleware* que se esté utilizando dentro de la infraestructura en Grid para el envío, puesta en marcha de la ejecución, monitorización y demás funcionalidades. Cuando se haya completado la ejecución de la tarea, se pedirá al mismo un informe de ejecución que será el que le sirva a SQS para aprender de sus éxitos o sus errores.

El informe de ejecución va a proporcionar datos reales donde en su momento, SQS, sólo pudo realizar una hipótesis. Comparar resultados hipotéticos con reales no será suficiente para evaluar la calidad del servicio y la satisfacción de cliente y proveedor. Es necesario, por tanto, utilizar reglas de negocio que definan qué objetivos pretenden optimizar tanto cliente como proveedor de servicios. A la luz del análisis de todos estos datos, SQS ya podrá determinar si su hipótesis fue lo suficientemente buena para cumplir los objetivos de cliente y de proveedor de servicios. De esta manera, podrá actualizar su sistema de toma de decisiones neuro-difuso de manera que la siguiente planificación sea mucho más óptima.

El *framework* de SQS tiene tres pilares básicos: su arquitectura bien definida que le permite una integrarse de manera sencilla con cualquier *middleware* de infraestructuras Grid; su extensibilidad para adoptar nuevas métricas e indicadores que ayuden a configurar cualquier tipo de clasificación de tareas y recursos; y su facilidad para definir las reglas del negocio para clientes y de proveedores de servicios.

En el siguiente apartado, se explica con todo lujo de detalles la arquitectura y diseño de los componentes que componen el *framework* SQS.

4. Arquitectura de SQS

Una de las principales características del *framework* SQS es su enorme sencillez arquitectónica. Como puede verse en la figura 2.8, SQS dispone básicamente de cuatro componentes: dos de ellos se encargan de la clasificación de las tareas (`NeuroFuzzyJobClassifier`) y de los recursos (`NeuroFuzzyResourceClassifier`) mientras que los otros dos se corresponden con el planificador basado en *soft-computing* (`SoftcomputingScheduler`) y el encargado de hacer aprender a los clasificadores tanto de tareas como de recursos.

Cada uno de los elementos que componen el sistema distribuido en Grid va a tener asociado un clasificador neuro-difuso de recursos (`NeuroFuzzyResourceClassifier`). Este clasificador, primero va a convertir los valores de las métricas elegidas en valores difusos en términos de pertenencia a cada uno de los conjuntos borrosos con los que se va a trabajar (***math***, ***render*** y ***webapp***).

Por otra parte, en el lado del usuario también va a existir un clasificador para sus tareas (`NeuroFuzzyJobClassifer`). Este clasificador, utilizará los indicadores elegidos para la definición de los trabajos y clasificará la tarea según su grado de pertenencia a cada uno de los conjuntos borrosos anteriormente mencionados (***math***, ***render*** y ***webapp***).

El responsable de hacer que SQS aprenda (`Teacher`) va a utilizar los datos que se tenían a priori y que sirvieron para realizar la hipótesis de donde ejecutar la tarea y los datos reales tras la ejecución. Estos datos junto con las reglas de negocio para clientes y proveedores van a determinar si la hipótesis fue lo suficientemente buena o no y por tanto determinarán el éxito o el fracaso de la planificación.

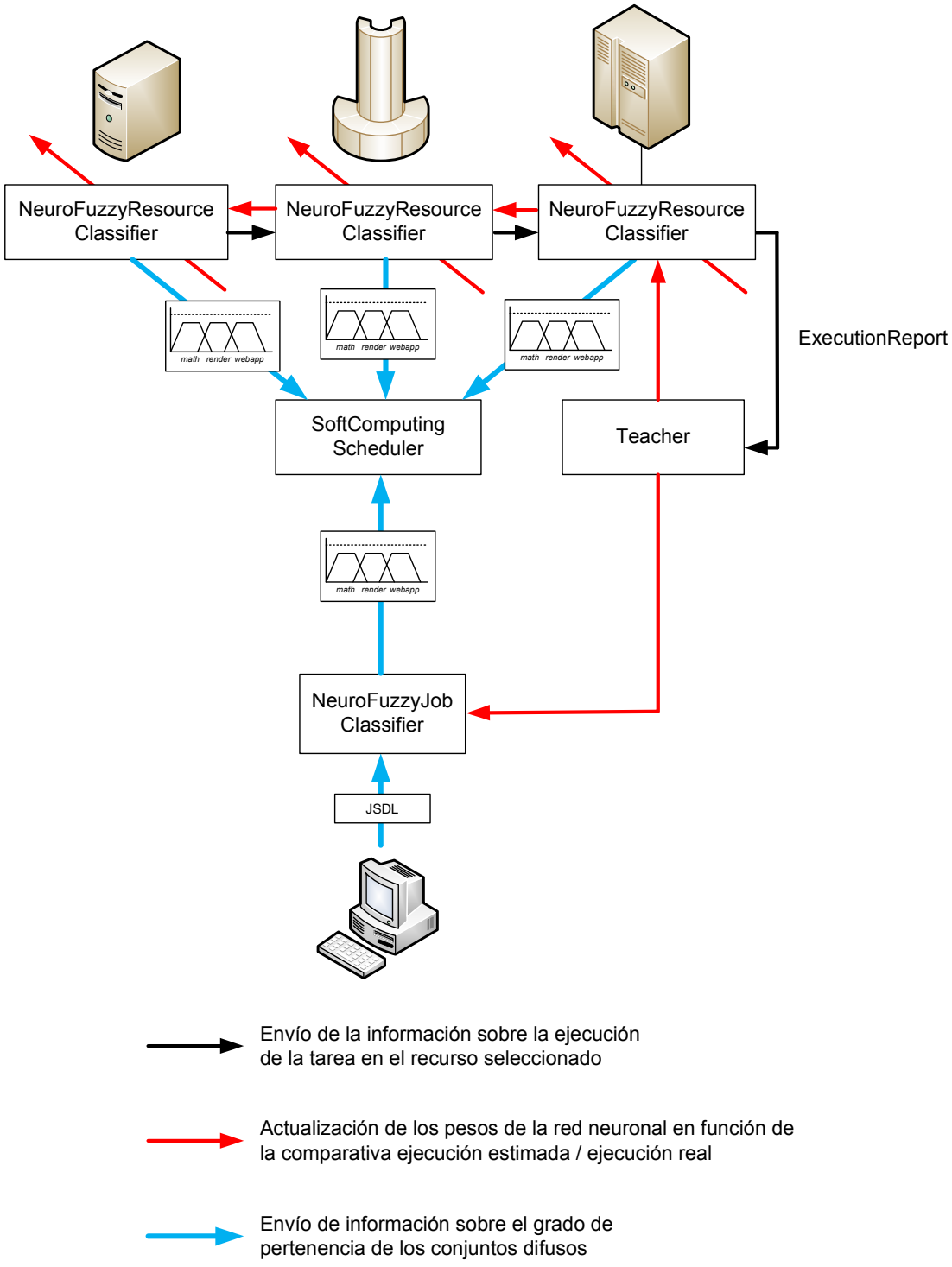


Figura 2.8. *Arquitectura de SQS.*

5. Desarrollo de la Inteligencia Artificial de SQS

La Inteligencia Artificial que utiliza SQS radica principalmente en su habilidad para clasificar tareas y recursos en función de planificaciones (junto con sus posteriores ejecuciones) pasadas.

Desarrollar la Inteligencia Artificial de SQS consiste en desarrollar los clasificadores neuro-difusos para los recursos y las tareas. Los sistemas neuro-difusos son sistemas híbridos a caballo entre los sistemas expertos difusos y las redes neuronales. Ya en la sección correspondiente al estado del arte se vio las características de ambos sistemas por separado. También se revisaron trabajos donde se han utilizado de manera independiente, pero son escasos en los que se han empleado de forma conjunta.

Un sistema neuro-difuso consiste en una red neuronal con una configuración especial de sus distintas capas. La figura 2.9, muestra un esquema general de construcción de sistemas neuro-difusos.

A continuación se detallan los detalles de construcción y las características de cada una de estas capas.

CAPA DE BORROSIFICACIÓN

Esta capa es la encargada de realizar el proceso de *borrosificación* típico de cualquier sistema difuso. En la figura 2.9, se corresponde con el color azul. La *borrosificación* es el proceso por el que vamos a convertir los valores nítidos de las variables de entrada de la red neuronal en valores difusos. Se utiliza una neurona por cada variable de entrada y conjunto difuso que quiera definir. Como puede verse en la figura 2.10, cada conjunto difuso se define dentro de una neurona.

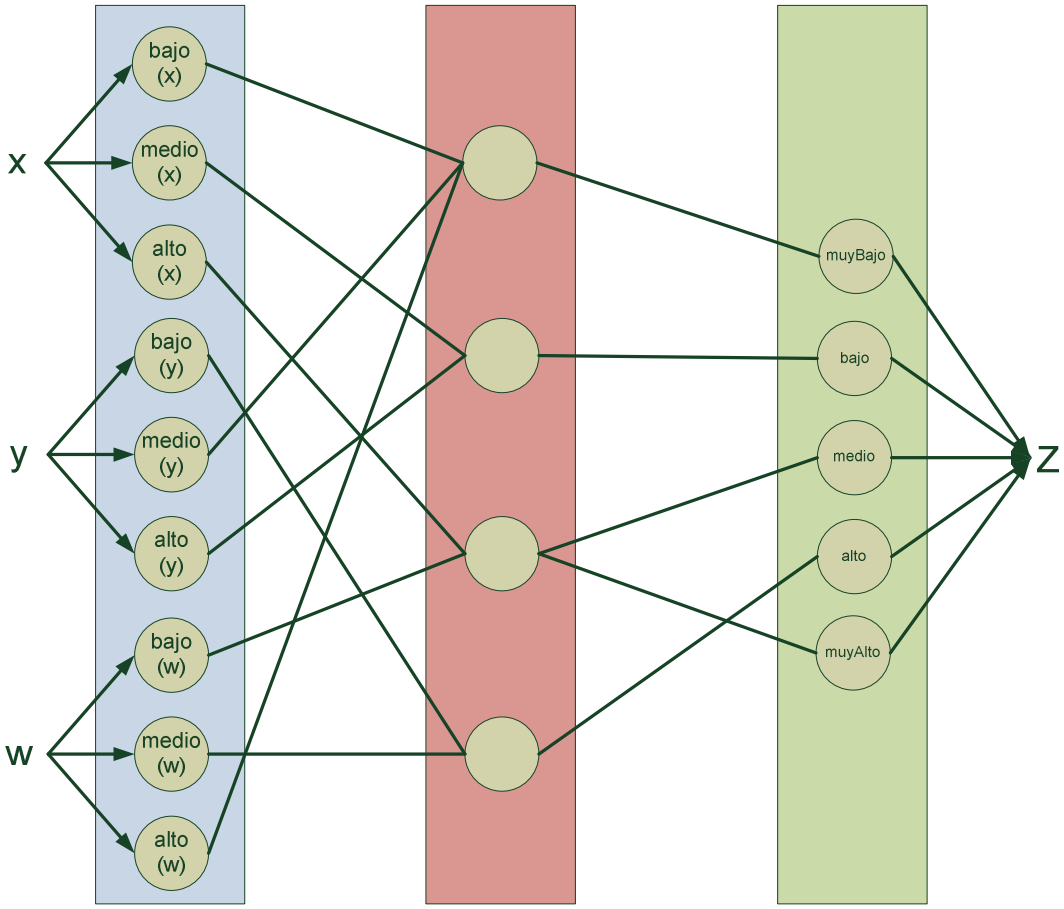


Figura 2.9. *Ejemplo de Sistema Neuro-Difuso.*

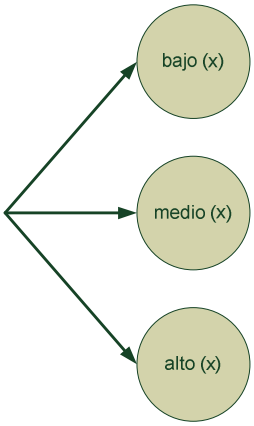


Figura 2.10. *Ejemplo de capa de borrosificación para un sistema neuro-difuso.*

CAPA DE DESBORROSIFICACIÓN

De forma simétrica a la capa de *borrosificación*, esta capa contiene los métodos clásicos de la lógica difusa con los que convertir valores difusos en nítidos.

En la figura 2.9, se corresponde con el color verde. En estas neuronas es donde hay que implementar alguno de los distintos métodos disponibles para la *desborrosificación*: método del centro de gravedad, método máx-min…

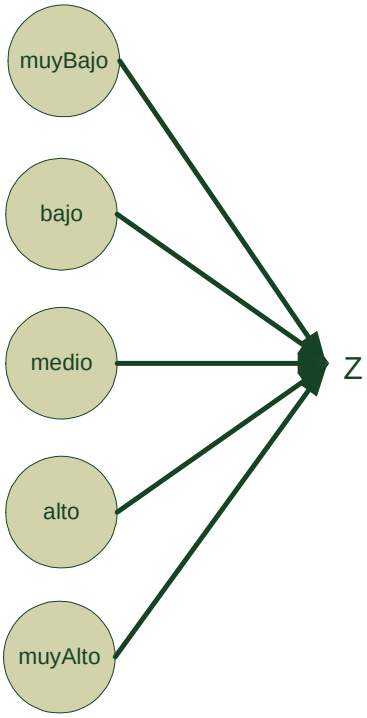


Figura 2.11. *Ejemplo de capa de desborrosificación para un sistema neuro-difuso.*

La figura 2.11 muestra el esquema de esta capa. El valor Z se corresponde con un valor numérico (nítido) resultado de la fase de *desborrosificación*.

CAPA DE REGLAS DIFUSAS

Esta capa es la más sencilla de construir. En la figura 2.9, se corresponde con el color rojo. Representa las reglas difusas de tipo *if…then* y para construirla únicamente hay que establecer conexiones sinápticas entre los antecedentes de las reglas difusas (*if*) que se encuentran en la capa de *borrosificación* y sus consecuentes (*then*) que se encuentra en la capa de *desborrosificación*. En el ejemplo de la figura 2.9, las conexiones sinápticas representan las siguientes reglas difusas:

```
if x==bajo & y==medio & w==alto then z==muyBajo

if x==medio & y==alto then z==bajo

if x==alto & w==bajo then z==medio & z==muyAlto

if y==bajo & w==medio then z==alto
```

DISEÑO DE UN CLASIFICADOR DE RECURSOS NEURO-DIFUSO (NEUROFUZZYRESOURCECLASSIFIER)

Una vez se ha presentado el método para la construcción de sistemas neuro-difusos de la sección anterior, pasaremos ahora al diseño del clasificador de recursos.

Variables de entrada

De entre las métricas que se vieron en la tabla 2.1 y para simplificar el diseño del clasificador de recursos, se han utilizado únicamente las siguientes tres:

- **CPU**: Que representa el porcentaje de uso de la CPU del recurso.
- **RAM**: Que representa el porcentaje de uso de memoria RAM del recurso.
- **RED**: Que representa el porcentaje de uso del Ancho de banda del recurso.

Capa de borrosificación

Lo primero que hay que hacer es definir qué variables lingüísticas que se van a utilizar para convertir los valores nítidos en difusos. En el caso del clasificador de

recursos, se ha elegido el conjunto {bajo, medio, alto} para definir los posibles estados de cada una de las variables de entrada, a saber, CPU, RAM y RED.

Para el proceso de *borrosificación* de las variables CPU, RAM y RED, una vez normalizadas en el intervalo [0, 1], se va a utilizar la función triangular que viene dada por las siguientes ecuaciones:

$$\text{bajo}(x) = \begin{cases} 1, & \text{si } x<0 \\ 1-2x, & \text{si } 0<=x<=0,5 \end{cases} \quad (1)$$

$$\text{medio}(x) = \begin{cases} 2x, & \text{si } 0<=x<0,5 \\ 2-2x, & \text{si } 0,5<=x<1 \end{cases} \quad (2)$$

$$\text{alto}(x) = \begin{cases} 2x-1, & \text{si } 0,5<=x<1 \\ 1-2x, & \text{si } x>1 \end{cases} \quad (3)$$

Estas ecuaciones producen los conjuntos borrosos de la figura 2.12.

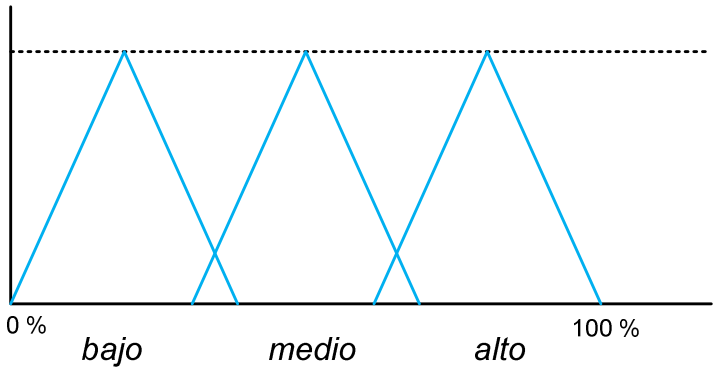


Figura 2.12. *Representación de las variables lingüísticas {bajo, medio, alto}*

Capa de desborrosificación

En el caso del clasificador de recursos, en esta capa no se pretenderá utilizar los métodos de conversión de valores difusos en nítidos. Estos datos difusos son los que utilizará el planificador para su comparativa contra los datos difusos de las tareas. No obstante, aquí se han de definir los conjuntos borrosos correspondientes a los

consecuentes de las reglas difusas. En nuestro caso, utilizaremos como variable lingüística `TipoTrabajo` siendo el resultado de la red neuro-difusa la pertenencia a cada uno de las clases de especialización que puede tener un recurso: ***math***, ***render*** y ***webapp***.

Capa de reglas difusas

Para el diseño de esta capa, lo primero que se ha de diseñar son el conjunto de reglas difusas que se van a utilizar. Merece la pena recordar que estas reglas difusas se utilizarán como si de un simple sistema experto difuso se tratase. Será a medida que avance el entrenamiento de la red neuronal cuando vayan cambiando los pesos de sus conexiones. Así, la activación de ciertas reglas van a tener asociado un peso mayor que otras, lo que representará el correspondiente entrenamiento de la red neuronal.

Las aplicaciones de simulación matemática son intensivas en uso de CPU y suelen utilizar moderadamente la memoria RAM del recurso ya que únicamente se utiliza para guardar resultados parciales. Además, el uso de la RED es muy bajo, ya que sólo se utiliza para el envío de los datos de entrada y el consecuente envío de los datos de salida al cliente.

Las aplicaciones de renderizado de imágenes son intensivas en CPU, aunque mucho menos que las anteriores, pero estas sí necesitan grandes cantidades de memoria RAM. Idealmente se desea que toda la información pueda mapearse en la memoria RAM para no tener que utilizar la memoria virtual ya que degrada el rendimiento debido a las operaciones de entrada/salida. En este caso, el uso de la RED también es moderado tirando a alto ya que los proyectos multimedia generan gran cantidad de datos que han de ser transmitidos del cliente al proveedor y viceversa.

Por último, las aplicaciones Web requieren un uso bajo de CPU, ya que suelen delegar en otros sistemas para la parte de procesamiento de la información, como pueden ser servidores de Bases de Datos. En cuanto a la memoria RAM, necesitan de un uso moderado tirando a bajo de la misma ya que únicamente necesitan mantener los objetos de los servidores de aplicaciones y crear eventualmente los objetos que representan cada una de las peticiones que se hacen a los servidores de aplicaciones. Sin embargo, el uso de

la RED es muy elevado ya que son sistemas que son utilizados por muchos usuarios generalmente y que estresan el sistema con constantes consultas.

Con esta definición de las aplicaciones en función de las variables de recursos que vamos a utilizar, se proponen las siguientes reglas difusas expertas:

```
if CPU=ALTO & RAM=MEDIO then TipoTrabajo=MATH
if CPU=MEDIO & RAM=ALTO & RED=BAJO then TipoTrabajo=RENDER
if RED=ALTO then TipoTrabajo=WEBAPP
```

Diseño final de la red neuro-difusa para el clasificador de recursos

La red neuronal que se ha utilizado en nuestro planificador neuro-difuso contiene tres capas como puede verse en la figura 2.13:

- **Capa de *borrosificación***, en azul, donde por cada variable lingüística (bajo, medio, alto), se ha utilizado una neurona que convierte en borrosa cada una de las entradas del sistema (CPU, RAM, RED).
- **Capa de *desborrosificación***, en verde, donde se convierten las variables lingüísticas de salida, en nuestro caso, `TipoTrabajo`, en un valor nítido utilizando alguno de los métodos de *Desborrosificación* conocidos. En el caso del planificador de recursos, no se va a utilizar esta capa ya que se utilizará el valor borroso la asignación de las tareas.
- **Capa de reglas difusas**, en rojo, donde mediante las conexiones sinápticas de la capa anterior se definen las reglas anteriormente citadas. Por ejemplo, para la regla `if RED=ALTO then TipoTrabajo=MATH` se han unido las neuronas `RED.ALTO` con `TipoTrabajo.MATH`.

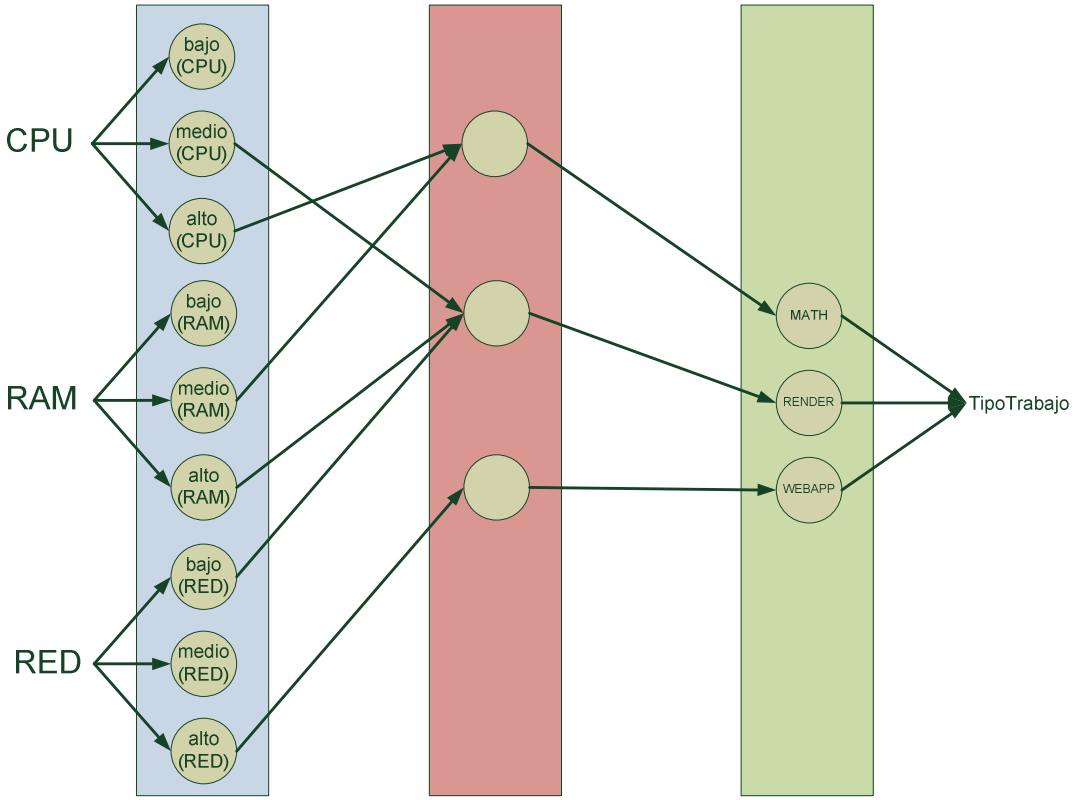


Figura 2.13. *Red neuro-difusa para el clasificador de recursos.*

DISEÑO DE UN CLASIFICADOR DE TRABAJOS NEURO-DIFUSO (NEUROFUZZYJOBCLASSIFIER)

Al igual que en el caso anterior para clasificar recursos, ya se justificó que, para las tareas, también se iba a utilizar un clasificador neuro-difuso. Siguiendo la misma estrategia de definición de dicho clasificador, se procede a continuación a definir con todo lujo de detalles las características de este otro clasificador:

Variables de entrada

Ya se comentó que la clasificación de trabajos iba a ser muy sencilla. Toda tarea dispone de información característica e independiente de las condiciones de ejecución de la misma. La intención de SQS de integrarse de forma sencilla con los *middleware* más utilizados, le hace pensar en utilizar el lenguaje JSDL. Utilizando algunos de los indicadores de JSDL y que un usuario puede rellenar, SQS va a proponer las siguientes variables de entrada para la clasificación:

- **TotalCPUTime**: El tiempo de CPU (en segundos) necesarios para ejecutar el trabajo.
- **TotalCPUCount**: La cantidad de CPUs necesarias para el trabajo.
- **TotalPhysicalMemory**: La cantidad (en *bytes*) de memoria RAM que necesita el trabajo.
- **TotalDiskSpace**: La cantidad (en *bytes*) de espacio en disco duro que necesita el trabajo.
- **Size**: El tamaño (en bytes) del trabajo.

Capa de Borrosificación

Al igual que en el caso de los recursos, la *borrosificación* va a convertir los valores nítidos de las variables de entrada en valores difusos.

Por simetría, se van a utilizar las mismas variables lingüísticas que en el caso de los recursos, a saber, ***bajo***, ***medio*** y ***alto***.

Para que sirvan las ecuaciones de la función triangular, hay que normalizar todas las variables en el intervalo [0, 1]. Para ello se proponen los valores máximos y mínimos de la tabla 2.3 para cada una de las métricas.

Tabla 2.3. *Normalización de los valores de entrada de las variables lingüísticas.*

Variable	Máximo	Mínimo
TotalCPUTime (GHz)	0	3
TotalCPUCount (núcleos)	0	8
TotalPhysicalMemory (GB)	0	8
TotalDiskSpace (TB)	0	1
Size (GB)	0	1

Con esta normalización, las variables lingüísticas vendrán dadas por las siguientes ecuaciones:

$$\text{bajo}(x) = \begin{cases} 1, & \text{si } x<0 \\ 1-2x, & \text{si } 0<=x<=0,5 \end{cases} \quad (1)$$

$$\text{medio}(x) = \begin{cases} 2x, & \text{si } 0<=x<0,5 \\ 2-2x, & \text{si } 0,5<=x<1 \end{cases} \quad (2)$$

$$\text{alto}(x) = \begin{cases} 2x-1, & \text{si } 0,5<=x<1 \\ 1-2x, & \text{si } x>1 \end{cases} \quad (3)$$

La definición de las variables lingüísticas utilizando estas ecuaciones, dependiendo del caso, generan las distribuciones de las figura 2.13. Se han representado con sus valores no normalizados para que sea más sencilla su lectura.

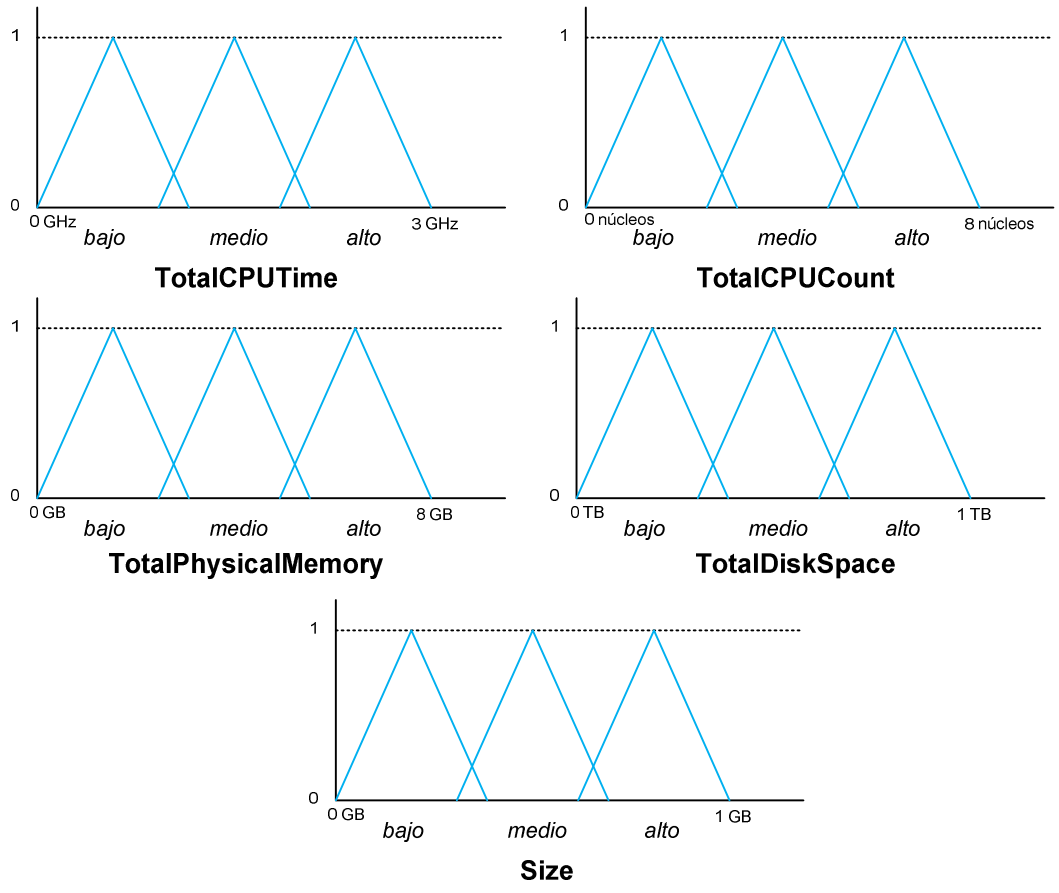


Figura 2.14. *Borrosificación de las variables de entrada.*

Capa de Desborrosificación

Con este proceso, se consigue que el valor difuso obtenido como resultado de aplicar las reglas difusas se convierta en un valor nítido (numérico) que podamos utilizar en el mismo contexto que utilizábamos con las variables de entrada. Aunque no nos interesa *desborrosificar* el valor ya que SQS va a trabajar con grados de pertenencia, aquí se han de definir los conjuntos borrosos correspondientes a los consecuentes de las reglas difusas. En nuestro caso, utilizaremos como variable lingüística `TipoTrabajo` siendo el resultado de la red neuro-difusa la pertenencia a cada uno de las clases de especialización que puede tener un recurso: ***math***, ***render*** y ***webapp***.

Capa de Reglas Difusas

Para que el clasificador neuro-difuso clasifique los trabajos según sean de trabajos matemáticos, de renderización o aplicaciones Web, se han de consultar, de igual forma que en el caso de la clasificación de recursos, a un experto que indique qué características han de tener cada una de las clases anteriores en función de la información que vamos a disponer de cada trabajo, es decir, en función del porcentaje de uso de sistema de ficheros, del sistema operativo, de la arquitectura de la CPU, del tiempo de CPU total, del número total de CPUs necesarias, de la cantidad de memoria física total, del espacio total en el disco y del tamaño del trabajo.

A partir de estas características de los trabajos, podemos inferir que si un trabajo tiene un tamaño alto y requiere mucho espacio en el disco duro, va a necesitar mucho Ancho de banda (RED) y por tanto la tarea va a ser de tipo ***webapp***.

También podemos inferir que si una tarea tiene un tiempo de CPU total bajo y el número de CPU son elevadas, sus necesidades de CPU serán muy altas y su perfil será de tipo ***math***.

De igual manera, si la cantidad de memoria física es elevada, su perfil será el de trabajo de tipo ***render***.

Con esta definición de las aplicaciones en función de las variables de trabajos que vamos a utilizar, se proponen las siguientes reglas difusas expertas:

```
if TotalCPUTime=BAJO & TotalCPUCount=ALTO then TipoTrabajo=MATH

if TotalPhysicalMemory=ALTO then TipoTrabajo=RENDER

if TotalDiskSpace=ALTO & Size=ALTO then TipoTrabajo=WEBAPP
```

Diseño final de la red neuro-difusa para el clasificador de trabajos

Como puede verse en la figura 2.15, de la construcción del clasificador neuro-difuso para los trabajos han aparecido las siguientes capas.

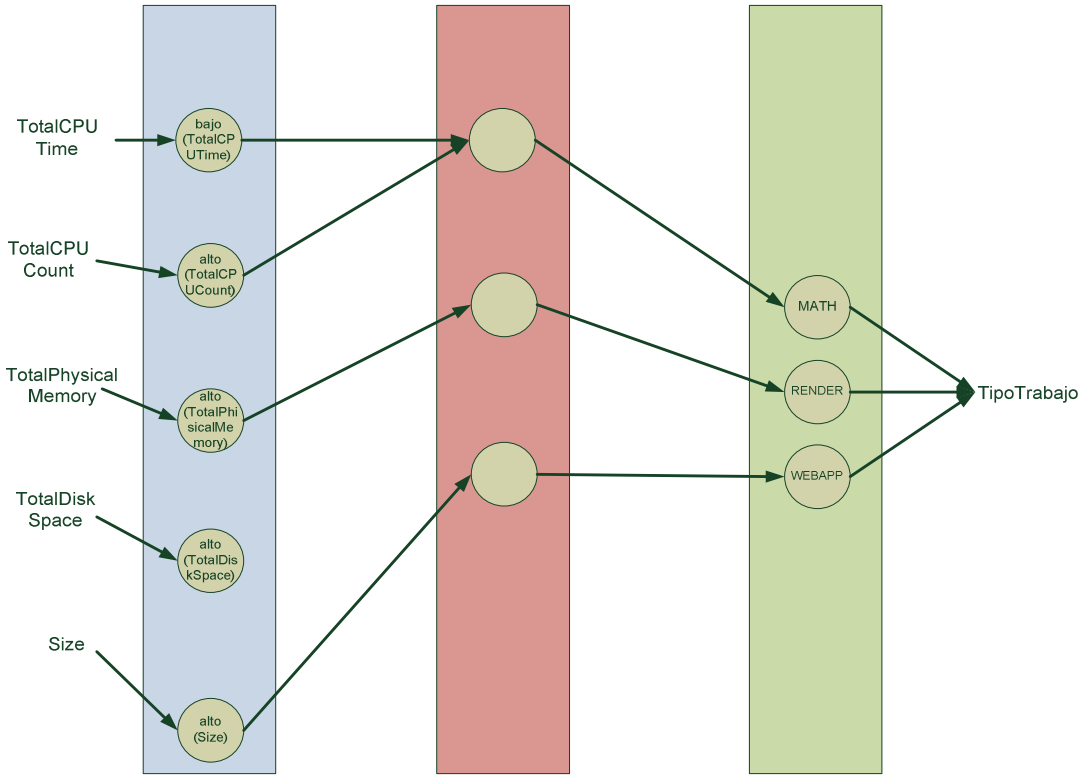


Figura 2.15. *Red neuro-difusa para el clasificador de tareas.*

Capa de *borrosificación*, en azul, donde por cada variable lingüística (bajo, medio, alto), se ha utilizado una neurona que convierte en borrosa cada una de las entradas del sistema (***TotalCPUTime***, ***TotalCPUCount***, ***TotalPhysicalMemory***,

TotalDiskSpace y ***Size***). Para simplificar la figura únicamente se han representado las neuronas responsable de la *borrosificación* que van a participar en las reglas difusas.

Capa de *desborrosificación*, en verde, donde se convierten las variables lingüísticas de salida, en nuestro caso, `TipoTrabajo`, en un valor nítido utilizando alguno de los métodos de *desborrosificación* conocidos. En el caso del planificador de recursos, no se va a utilizar esta capa ya que se utilizará el valor borroso la asignación de las tareas.

Capa de reglas difusas, en rojo, donde mediante las conexiones sinápticas de la capa anterior se definen las reglas anteriormente citadas. Por ejemplo, para la regla `if TotalPhysicalMemory=ALTO then TipoTrabajo=RENDER` se han unido las neuronas `TotalPhysicalMemory.ALTO` con `TipoTrabajo.RENDER`.

6. Toma de decisiones basado en conjuntos borrosos

Una vez se ha obtenido de los la clasificación del tarea, así como la especialización de cada uno de los recursos, llega el momento en que SQS tiene que tomar la decisión de qué planificación es la mejor que se puede obtener. Para tomar esta decisión, SQS va a utilizar un método totalmente novedoso basado en la comparación de conjuntos borrosos.

DISEÑO DEL PLANIFICADOR INTELIGENTE (SOFTCOMPUTINGSCHEDULER)

El Planificador Inteligente recibe de una parte el resultado de la clasificación de la tarea en función. De igual manera, consulta a todos los recursos por su especialidad en ejecución de tareas. En ambos casos, se obtiene el grado de pertenencia a las variables lingüísticas ***math***, ***render*** o ***webapp***.

Con todos estos datos, sólo hay que comparar los datos borrosos correspondientes a la tarea con los datos borrosos de cada uno de los recursos para ver a cual se ajusta mejor.

El *framework* SQS dispone de un mecanismo de comparación de la similitud entre tareas y recursos en función de la cual, será capaz de decidir a qué recurso enviar la tarea de forma óptima. Este mecanismo queda definido por los algoritmos que se detallan a continuación.

Algoritmo borroso para el cálculo de similitud entre tareas y recursos

Para garantizar la similitud entre la tarea y los recursos en términos borrosos es lo más parecido posible, se va solapar el resultado de la clasificación borrosa de cada tarea sobre cada uno de los recursos y se calculará el área de la intersección resultado de dicho solape. El recurso cuya intersección con la tarea sea menor, indicará que su ajuste es mayor.

La manera de realizar esta comparativa es muy sencilla. Debido a que tanto tareas como recursos pertenecen en determinado grado a cada uno de los conjuntos borrosos definidos, SQS propone el siguiente algoritmo:

```
para_cada r perteneciente_a {r1, r2, ..., rn}
      obtenerValoresBorrosos {math, render, webapp}
      para_cada cb perteneciente_a {math, render, webapp}
            areaSimilitud ← areaTarea(cb) ∩ areaRecurso(i, cb)
            tablaSimilitudes(r, cb) ← areaSimilitud
      fin_para_cada
fin_para_cada
```

Algoritmo 2.1. *Construcción de tabla de similitudes de recursos y tareas*

Con el algoritmo anterior, se va a obtener una tabla de similitudes de dimensión R x C donde R es el número de recursos y C representa el número de conjuntos borrosos con el que se trabaje.

Una vez disponemos de la tabla de similitudes, es momento de que SQS la utilice para decidir finalmente a qué recurso asignará la tarea.

Algoritmo de toma de decisiones a partir de una tabla de similitudes

La interpretación de la tabla de similitudes es clave dentro de la toma de decisiones de SQS. Los valores de la tabla representan el paso de *desborrosificación* que no se realizó en cada una de las redes neuro-borrosas. En efecto, cada uno de los valores de la tabla representa la similitud de tareas y recursos de forma que, cuanto menor sea ese valor, mayor será la similitud entre ellas.

Así, utilizando la tabla, se propone el siguiente algoritmo para decidir qué recurso va a ser el elegido para la tarea en cuestión:

```
recurso_seleccionado ← r1
areaMinima ← 0
para_cada r perteneciente_a {r1, r2, ..., rn}
      para_cada cb perteneciente_a {math, render, webapp}
            si tablaSimilitudes(r, cb) < areaMinima
                  recurso_seleccionado ← r
      fin_para_cada
fin_para_cada
```

Algoritmo 2.2. *Selección del recurso más apropiado para la tarea.*

La forma en que se combinan los dos algoritmos anteriormente explicados para conseguir implementar la toma de decisiones, puede verse en la figura 2.16. En esta figura puede verse cómo, a partir de las entradas difusas que le llegan al planificador, se obtiene utilizando los dos algoritmos, el recurso seleccionado.

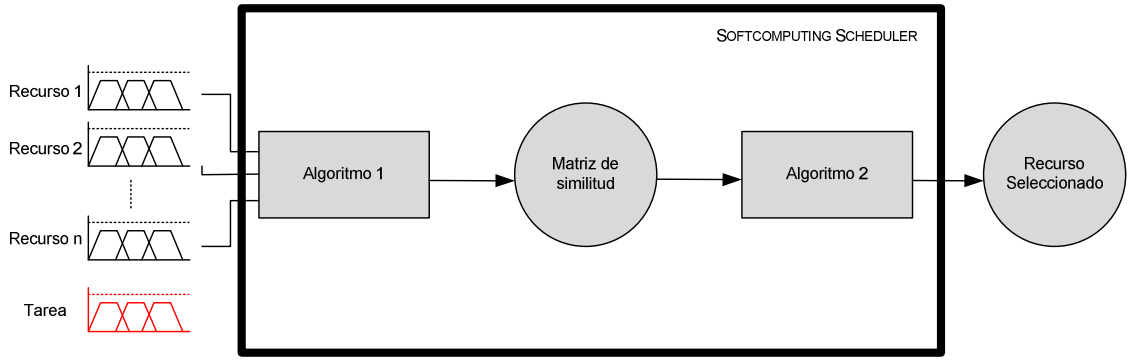


***Figura 2.16**. Arquitectura del Planificador Inteligente.*

Para ilustrar mejor los algoritmos con el que se implementa la toma de decisiones, se presenta a continuación un ejemplo.

El ejemplo comienza cuando el planificador ha recibido la clasificación borrosa de la tarea así como la de todos los recursos disponibles en el sistema en Grid.

Ejemplo

Supongamos que disponemos de tres recursos, como se mostró en la arquitectura de la figura 2.8.

La figura 2.17 muestra de manera gráfica el resultado de intersecar el grado de pertenencia a cada uno de los conjuntos borrosos (***math***, ***render*** y ***webapp***) tanto de la tarea como del primer recurso. En esta figura puede verse cómo el área más pequeña corresponde al área amarilla que se corresponde con el conjunto borroso ***render***. La interpretación que SQS hace a la luz de esta gráfica es:

"Creo que esta tarea es de tipo render y ha de ejecutarse en el Recurso 1 que está suficientemente especializado en dichas tareas en este momento".

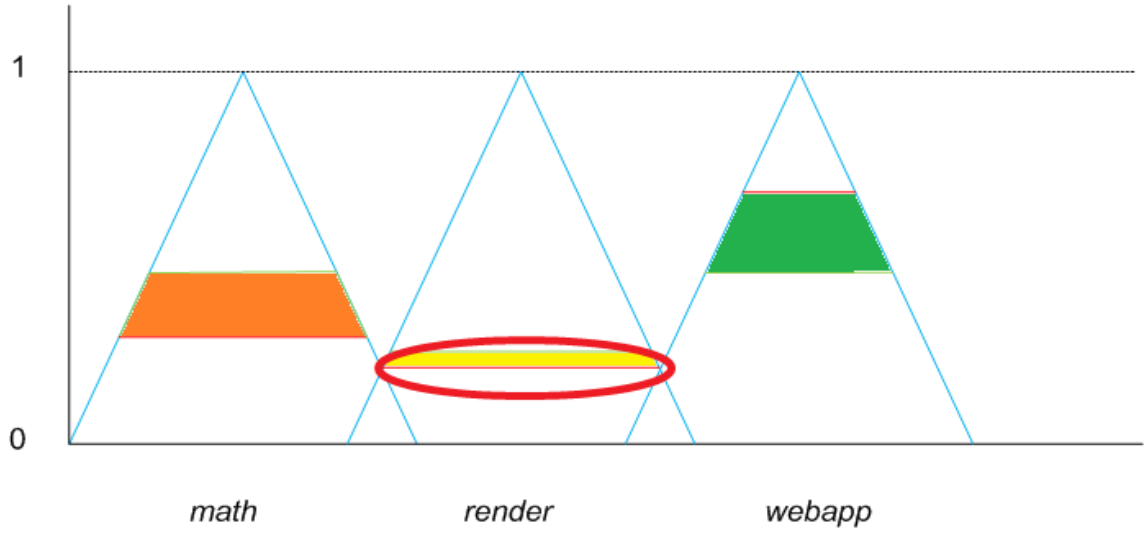


Figura 2.17. *Intersección de variables borrosas de la Tarea y el Recurso 1.*

Por su parte, la figura 2.18, muestra la intersección de los conjuntos borrosos de la tarea clasificada con el segundo candidato a ejecutarla: el Recurso 2.

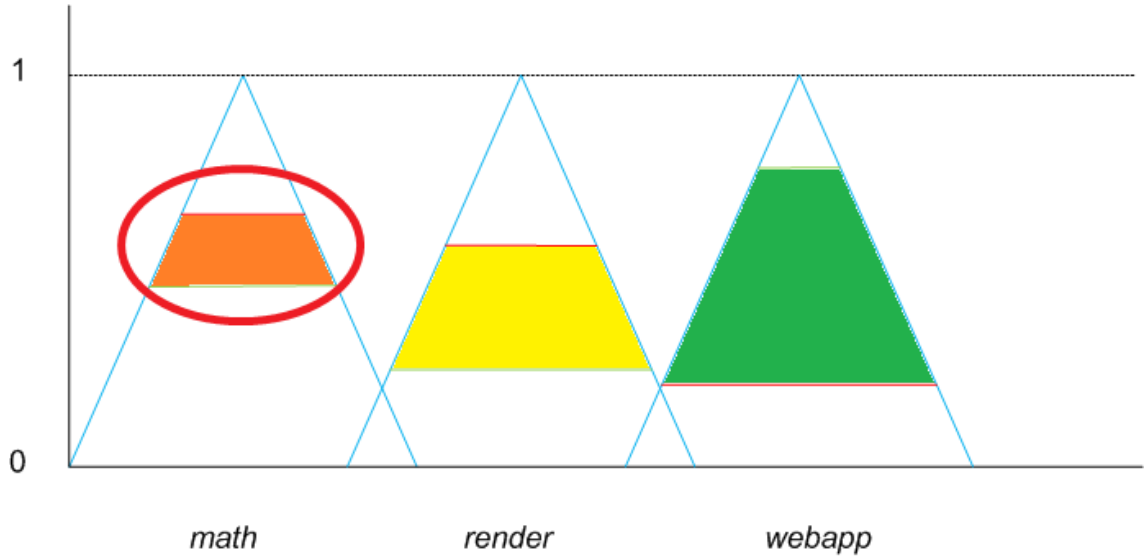


Figura 2.18. *Intersección de las variables lingüísticas de la Tarea y el Recurso 2.*

En esta figura, puede verse cómo el área más pequeña corresponde el área naranja que se corresponde con el conjunto borroso ***math***. La interpretación que SQS hace a la luz de esta gráfica es:

"Creo que esta tarea es de tipo math y ha de ejecutarse en el Recurso 2 que está suficientemente especializado en dichas tareas en este momento".

Por último, la figura 2.19 muestra la intersección de la tarea, con el Recurso 3: el último candidato a ejecutar la tarea. con el Recurso 3: el último candidato a ejecutar la tarea.

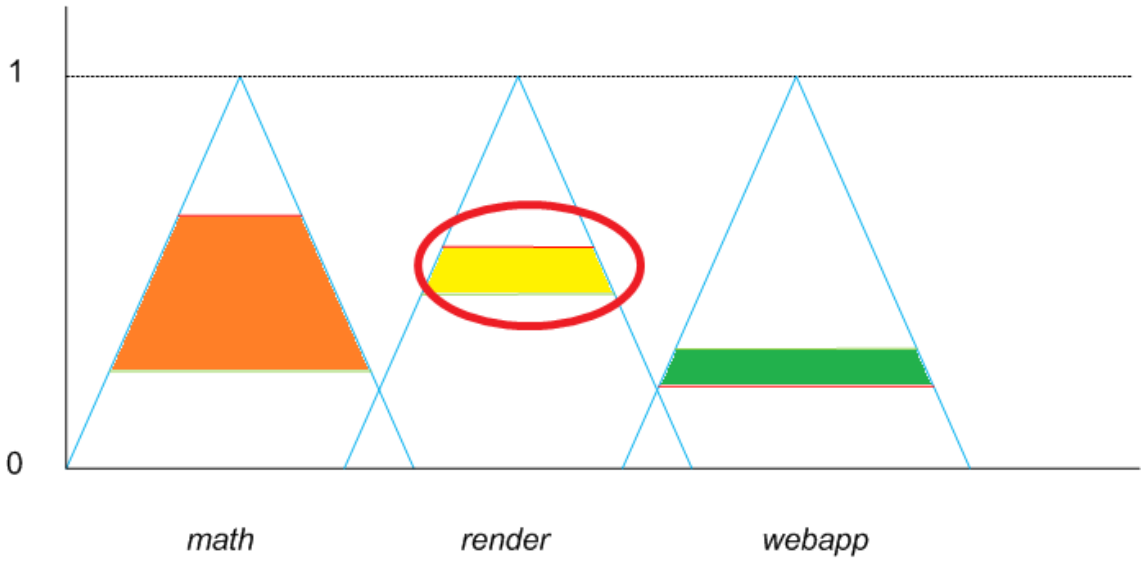


Figura 2.19. *Intersección de las variables lingüísticas de la Tarea y el Recurso 3.*

En este caso, una vez más gana el área de color amarillo, lo que hace pensar a SQS lo siguiente:

"Creo que esta tarea es de tipo render y ha de ejecutarse en el Recurso 3 que está suficientemente especializado en dichas tareas en este momento".

Una vez calculadas, todas las áreas, la tabla 2.4 representa la tabla de similitudes.

Tabla 2.4. *Tabla de similitudes.*

	Tarea		
	math	*render*	*weabpp*
Recurso 1	35	**10**	65
Recurso 2	20	30	85
Recurso 3	50	20	30

S e ha resaltado en negrita el valor más bajo de todos que se corresponde con la decisión por parte de SQS de enviar la tarea al Recurso 1 bajo la hipótesis de que ambos están en estado ***render***.

Llegado a este momento, SQS tiene que elegir entre tres opciones:

- Puede mandar el trabajo al Recurso 1 suponiendo que tanto la tarea como el recurso son de tipo ***render***
- Puede mandar el trabajo al Recurso 2 suponiendo que ambos son de tipo ***math***.
- Puede mandar el trabajo al Recurso 3 pensando que ambos son de tipo ***render***.

Al ser el área más pequeña la que comparte la Tarea con el Recurso 1, será a este a donde la envíe para su ejecución.

Interpretación de los resultados

Esta decisión puede sorprender al lector ya que el grado de pertenencia tanto de la tarea como del recurso al tipo ***render*** son los más bajos para ese caso. Hay que recordar que lo que está haciendo aquí SQS es una hipótesis basada en características ocultas de tareas y recursos. Hasta no ejecutar la tarea en ese recurso y obtener el informe de resultados no deberá el lector suponer que la planificación ha sido incorrecta.

No obstante, el método de selección es muy robusto incluso desde el punto de vista de la *desborrosificación*. Nótese que no se está utilizando el área mínima entre Tarea y Recurso que representaría el método de *desborrosificación* del mínimo, sino que se está utilizando el área encerrada entre grados de pertenencia a cada uno de los conjuntos borrosos. De esta manera, cuanto más pequeña esta área, más similares serán tarea y recurso pero no sólo eso.

Por otra parte, cuanto más elevado sea el grado de pertenencia de la Tarea y el Recurso de forma conjunta a un conjunto borroso dado, será también menor su área. Esto significará que tanto Tarea como Recurso están más cerca del valor 1 que representa en pertenecer totalmente a dicho conjunto borroso.

Si combinamos ambas métricas estamos consiguiendo maximizar la similitud de Tarea y Recurso así como el grado de pertenencia a un conjunto borroso dado, lo que justifica que el método es óptimo.

7. Estrategia de aprendizaje para el clasificador neuro-difuso

El método de aprendizaje que vamos a utilizar en nuestro clasificador neuro-difuso tanto de trabajos como de recursos va a ser una combinación de aprendizaje supervisado en línea y aprendizaje competitivo.

El **aprendizaje supervisado en línea** se fundamenta en disponer, a priori, de entradas y salidas conocidas. De tal manera, podemos establecer un umbral de error que nos sirva como discriminante para hacer que el aprendizaje sea positivo o negativo. Además, cuando es de tipo en línea (*online*) lo que ocurre es que no se dispone de los ejemplos de entrenamiento a priori, sino que se van obteniendo sobre la marcha.

Por otra parte, el **aprendizaje competitivo** se basa en que sólo quede un camino dentro de la red neuronal que produzca activación de las neuronas y el resto queden inhibidas.

En el caso de SQS, el aprendizaje se producirá cada vez que dispongamos del informe de resultados de la ejecución de la tarea sobre el recurso asignado. Al ser la red neuro-difusa de tipo en línea, su entrenamiento se producirá cada vez que se obtenga este informe de resultados y se pueda comparar con la hipótesis realizada y las reglas de negocio impuestas por cliente y proveedor de servicios.

Al combinar el aprendizaje anterior con el aprendizaje competitivo, a medida que la red neuro-difusa aprenda, irá ajustando los pesos de las conexiones sinápticas de manera que la importancia de cada una de las reglas tienda a converger hacia una solución óptima. Finalmente, sólo un conjunto de reglas difusas van a activarse lo que indicará que son las correctas a la hora de clasificar tareas o recursos.

APRENDIZAJE DE LA RED NEURO-DIFUSA (TEACHER)

Una vez se ha completado el trabajo en el recurso seleccionado, se dispondrá del informe de ejecución con los valores reales que, en la descripción del trabajo (dentro del archivo JSDL) eran hipotéticos y estimados por el usuario.

Las redes neuro-difusas tienen la capacidad de aprender y el método elegido para que este aprendizaje se produzca será clave para SQS. La forma en que las redes neuro-borrosas aprenden es mediante la modificación de los pesos de sus conexiones sinápticas. En el caso de que el aprendizaje sea positivo, los pesos de las conexiones sinápticas se incrementa, mientras que en caso de que el aprendizaje sea negativo, los pesos se decrementan.

Para poder decidir si el aprendizaje ha de ser positivo o negativo, se necesitan tres elementos:

- La **hipótesis de asignación** de la tarea en un recurso
- El **informe de ejecución**, donde se van a encontrar todos los datos relevantes de la ejecución de la tarea en el recurso.
- Las **reglas de negocio**, que servirán al Profesor para comparar cuantitativamente la ejecución real frente a la hipótesis realizada. Gracias al *framework* SQS, las reglas de negocio son totalmente configurables, personalizables y extensibles. Estas reglas de negocio van a representar los aspectos que realmente importan tanto a los usuarios como a los proveedores de servicios. Dentro de estas reglas de negocio se van a poder incluir restricciones temporales en cuanto al tiempo máximo en que una tarea se termine, en cuanto a que los recursos de un proveedor de servicios estén permanentemente ocupados, que se maximice el beneficio económico, que no se pierda dinero siempre que sea posible, esperar a que llegue un cliente con un beneficio fijo determinado… Todas estas reglas de negocio dependerán de la estrategia de empresa y clientes a la hora de utilizar los sistemas distribuidos en Grid desde el punto de vista del *Utility computing*.

  La arquitectura del sistema de aprendizaje completo puede verse en la figura 2.20.

El aprendizaje comparará datos reales con datos hipotéticos en función de las reglas de negocio definidas. Así se averigua si la hipótesis de enviar ese trabajo a ese recurso fue suficientemente aceptable o no. Por su parte, SQS decidirá, con esos datos, si el aprendizaje ha de ser positivo o negativo. Esta conclusión servirá para hacer aprender tanto a los clasificadores de recursos, como al clasificador de tareas actualizando los pesos de sus conexiones sinápticas.

De esta manera, se conseguirá que, a medida que se vayan realizando hipótesis y se vayan contrastando los resultados reales con ellas, la activación de unas y otras reglas difusas de los clasificadores neuro-difusos vayan tomando más o menos relevancia.

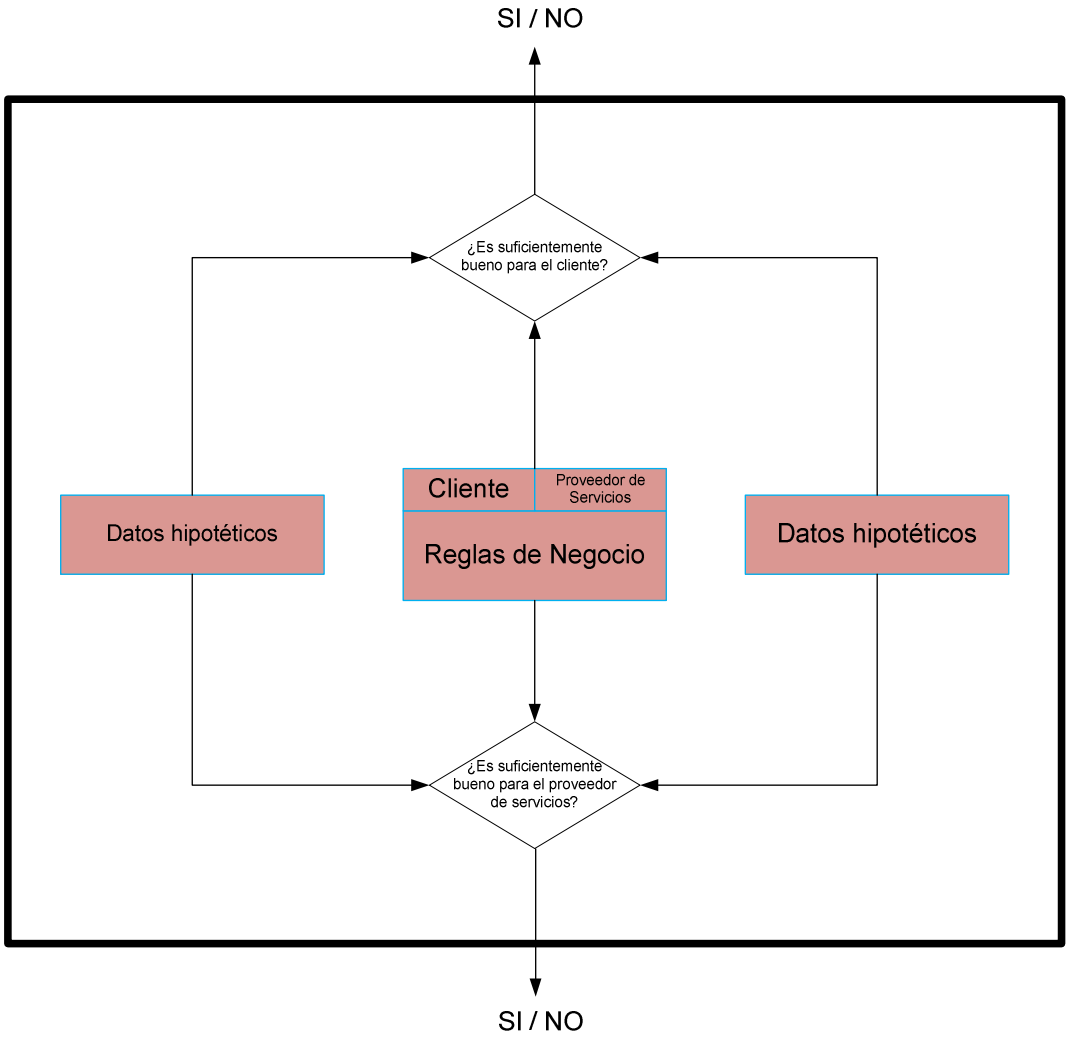


Figura 2.20. *Profesor de SQS - Responsable de su aprendizaje.*

3. CONCLUSIONES Y TRABAJO FUTURO

El problema de planificación en sistemas distribuidos en Grid necesita resolver de manera inteligente el problema de planificación de tareas en recursos. Las técnicas de Inteligencia Artificial propuestas hasta ahora, se han centrado en la resolución de este problema desde el punto de vista clásico de la logística y la investigación operativa.

El *framework* SQS ofrece la potencia de una solución distribuida, tolerante a fallos, escalable y eficiente que ofrecen los sistemas distribuidos en Grid, junto con las ventajas de las técnicas de Inteligencia Artificial incluidas dentro de la vertiente *soft-computing*.

Se pretende con su propuesta, introducir la Inteligencia Artificial en la planificación de sistemas distribuidos en Grid desde una perspectiva innovadora y novedosa lejos de las propuestas de la literatura revisada y especializada en el tema. En concreto se pretende abandonar el camino de las técnicas clásicas de planificación y enfocarla desde el punto de vista de la clasificación de tareas y la especialización de recursos.

El uso de las otras dos técnicas de *soft-computing*, a saber, lógica difusa y redes neuronales en combinación para formar un sistema neuro-difuso ofrecen las ventajas de trabajar con información imprecisa a la par que un método de aprendizaje en línea que se alimenta cada vez que se realiza una nueva planificación.

1. Principales aportaciones

Las principales aportaciones que se concluyen del desarrollo de este trabajo son:

- Diseño de un *framework* extensible para la creación de planificadores neuro-difusos basados en la clasificación de los trabajos que llegan a un planificador de entornos distribuidos en Grid.
- Diseño de una red neuro-difusa para la clasificación de los trabajos en función de métricas inherentes a los trabajos como su tamaño, la arquitectura donde deben ejecutarse, el sistema operativo sobre el que ejecutarse, así como variables relacionadas con la satisfacción del cliente como el presupuesto que tiene para la ejecución de dicha tarea, el tiempo máximo del que dispone antes de necesitar el resultado…
- Diseño de una red neuro-difusa para la especialización de los recursos del sistema distribuido en Grid. Se han utilizado para esta clasificación métricas relacionadas con el porcentaje de uso de la CPU, de la memoria RAM, del ancho de banda… así como variables relacionadas con la satisfacción del proveedor de los servicios a fin de maximizar la satisfacción del mismo en términos de obtener máximo beneficio o mantener sus recursos el máximo tiempo posible ocupados.
- Diseño de algoritmos borrosos para la toma de decisiones en base a los grados de pertenencia de trabajos y recursos a los distintos conjuntos difusos definidos. Esta toma de decisiones elije el recurso para el que él y la tarea son más parecidos en términos borrosos.

2. Trabajo futuro

El trabajo iniciado con la realización de este proyecto de investigación y que verá su continuación en forma de tesis doctoral propone desarrollarse en cuatro líneas de trabajo.

La primera consiste en potenciar el *framework* SQS mediante el uso de algoritmos evolutivos para la simulación de tareas de diferentes características en recursos que se encuentren en estados totalmente distintos y heterogéneos. Alimentando con estos datos

no oficiales pero si oficiosos a la red neuronal, se conseguirá aumentar el número de ejemplos de entrenamiento y, por tanto, aumentar la eficiencia de los resultados que origine. Aquí la integración con entornos de simulación de sistemas distribuidos en Grid puede ser de gran interés.

Una segunda línea de trabajo pasa por hacer una extensión de las reglas de negocio para la maximización del beneficio de clientes y proveedores. Construir un lenguaje de reglas de negocio sencillas para individuos no relacionados con las tecnologías de la información abriría una nueva ventana en las interfaces entre humanos y máquinas. Para conseguir este objetivo, se pretende trabajar en la línea de análisis de textos manuscritos y de contratos de acuerdos de prestación de servicios (SLA - *Service Level Agreement*) a partir de los cuales pueda obtenerse información relevante y probablemente oculta o difícil de representar en reglas que se apliquen a las reglas de optimización de la planificación.

En tercer lugar, dado que el aprendizaje realizado por el sistema neuro-difuso sigue el patrón del ciclo CBR (*Case-Based Reasoning*), otra de las líneas de trabajo futuras consistirá en utilizar un sistema CBR donde se almacenen los casos de éxito y fracaso de planificación de tareas sobre recursos, de manera que para asignaciones presentes y futuras puedan ajustarse por interpolación o por los métodos definidos en la fase de adaptación. En esta línea, se propone la integración con *jColibri* que se ha convertido en uno de los *framework* de razonamiento basado en casos en Java más utilizados.

Por último, pero no por eso menos importante, y como integración en sistemas de gestión de recursos distribuidos en Grid, la última línea de trabajos futuros consistiría en integrar este planificador dentro de los *framework* más utilizados de manera que puedan alimentarse de la potencia de una planificación inteligente. En esta línea sería de gran interés integrar a SQS en la planificación de trabajos que realiza el meta-planificador de *GridWay* proponiendo un método totalmente innovador hasta la fecha en cuanto a los métodos de planificación para sistemas distribuidos en Grid.

Anexo I. HERRAMIENTAS SOFTWARE DISPONIBLES

I. Herramientas para sistemas distribuidos en Grid

Como *framework* de desarrollo para entornos Grid, disponemos del estándar de facto en que se ha convertido Globus Toolkit 4 (Foster, 2005)

No obstante, aun siendo el estándar de facto, es muy tedioso desarrollar ciertas tareas como pueden ser enviar un trabajo al necesitar muchas líneas de código para ello (Yu, Luo, Chou, Chen, & Zhou, 2007).

Además, no dispone de un algoritmo de planificación adecuado para que el usuario pueda saber qué recurso es el mejor para su trabajo (eso sigue siendo cierto hoy??? No utiliza GridWay???) (Yu, Luo, Chou, Chen, & Zhou, 2007)

La necesidad de un entorno sencillo en el que desarrollar las pruebas, nos hacen considerar simuladores en vez de entornos Grid reales, al menos para la evaluación preliminar de los resultados.

Para la evaluación de los resultados, hay a nuestra disponibilidad distintos simuladores con los que obtener resultados aceptables antes de llevar las propuestas a un entorno de producción Grid real.

El caso de (Yu, Luo, Chou, Chen, & Zhou, 2007) utiliza NAS Grid Benchmarking para evaluar su propuesta

En (LaTorre, Peña, Robles, & de Miguel, 2008), se hace una revisión de herramientas útiles para la evaluación de los distintos problemas de planificación de supercomputación planteados.

- **La familia de Gestores de Recursos PBS** (*PBS*, *OpenPBS*, y *Torque*) implementan un planificador *First-Come-First-Served*, pero también proporcionan mecanismos para implementar otras planificaciones.
- **El Planificador de Clúster Maui** es compatible con el Gestor de Recursos Torque. Proporciona distintas políticas de planificación para acomodar las diferentes necesidades de planificación.
- **SLURM (*Simple Linux Utility for Resource Management*)** (Yoo, Jette, & Grondona, 2003) es un Gestor de Recursos de código libre tolerante a fallos de y alta escalabilidad para la planificación de trabajos en clúster Linux de cientos de nodos. Entre sus componentes existe uno para la gestión del estado de las máquinas, para la gestión de las particiones, para la gestión de los trabajos, para la planificación y para los módulos de copia de flujos de datos.
- En cuanto a los productos comerciales, el más conocido es **LoadLeveler** (Kannan, Roberts, Mayes, Brelsford, & Skovira, 2001) de IBM que proporciona una gestión básica de colas basada en prioridades. Estas prioridades pueden configurarse en función de distintas políticas.

Uno de los simuladores de entornos Grid que resultan interesantes es **GridSim** (Sulistio, Cibej, Venugopal, Robic, & Buyya, 2008). Por ejemplo, se utiliza en (Jia, Rajkumar, & Kotagiri, 2008) para el estudio realizado.

Otro *framework* utilizado para la simulación de entornos Grid es **Hypersim** (Xhafa, Alba, Dorronsoro, Duran, & Abraham, 2008) que es un planificador centralizado que evita el problema de las interacciones entre los distintos planificadores locales. En el simulador no se ha considerado la posibilidad de dependencias entre recursos y entre tareas.

II. Herramientas para la creación de Redes Neuronales

Las dos herramientas más utilizadas para la creación y uso de Redes Neuronales en Java son Joone (Java Object Oriented Neural Engine) y Neuroph (Sevarac).

Joone es un *framework* para Redes Neuronales escrito en Java. Su componente principal es el motor de redes neuronales y le acompaña un editor GUI y un entorno de entrenamiento distribuido que puede extenderse escribiendo nuevos módulos para implementar nuevos algoritmos o arquitecturas a partir de sus componentes iniciales.

Neuroph es un *framework* para Redes Neuronales ligero. Contiene una librería bien diseñada y de software libre en Java con un número reducido de clases básicas que se corresponden con los conceptos de Redes Neuronales básicos. También dispone de un entorno gráfico con el que crear rápidamente componentes.

III. Herramientas para entornos de Lógica Difusa

Entre las herramientas más interesantes para el desarrollo de sistemas que utilicen lógica difusa se destacan XFuzzy (Moreno Velo, Baturone, Sánchez Solano, & Barriga, 2001) y JFuzzyLogic (Cingolani).

XFuzzy es un *framework* para sistemas expertos basados en lógica difusa que proporciona distintas herramientas con las que completar el proceso de desarrollo de este tipo de sistemas. Sus principales características es el poder desarrollar sistemas complejos y disponer de herramientas con las que el usuario puede extender y desarrollar nuevas funcionalidades en base a dicho *framework.*

JFuzzyLogic es un paquete de lógica difusa escrito en Java que implementa la especificación IEC 1131p7 de lenguaje de control difuso FCL (*Fuzzy Control Language*). Utilizar el lenguaje FCL libera al usuario de tener que conocer las clases de implementación en Java con las que codificar las reglas difusas y puede limitarse a escribir un sencillo archivo de texto donde definirlas utilizando un lenguaje de reglas *if…then*. La figura I.1, muestra un ejemplo un conjunto de reglas difusas en lenguaje FCL.

```
RULEBLOCK No1
    AND : MIN;
    // Use 'min' activation method
    ACT : MIN;
    // Use 'max' accumulation method
    ACCU : MAX;

    RULE 1 : IF cpu IS low AND ram IS low THEN systemLoad IS veryLow;
    RULE 2 : IF cpu IS low AND ram IS medium THEN systemLoad IS medium;
    RULE 3 : IF cpu IS low AND ram IS high THEN systemLoad IS high;
    RULE 4 : IF cpu IS medium AND ram IS low THEN systemLoad IS low;
    RULE 5 : IF cpu IS medium AND ram IS medium THEN systemLoad IS medium;
    RULE 6 : IF cpu IS medium AND ram IS high THEN systemLoad IS high;
    RULE 7 : IF cpu IS high AND ram IS low THEN systemLoad IS medium;
    RULE 8 : IF cpu IS high AND ram IS medium THEN systemLoad IS high;
    RULE 9 : IF cpu IS high AND ram IS high THEN systemLoad IS veryHigh;

END_RULEBLOCK
```

Figura I.1. *Ejemplo de definición de reglas difusas en lenguaje FCL.*

PUBLICACIONES

La realización de este trabajo ha producido la siguiente publicación:

Tarancón, S., Garmendia, L. & Santos, M. (2009). Red Neuronal Difusa para un Planificador de Entornos Grid. *Jornadas internacionales de Didáctica de las Matemáticas en Ingeniera. E.T.S.I. Caminos, UPM*, (págs. 313-320). Madrid.

BIBLIOGRAFÍA

Ali, S., Siegel, H., Maheswaran, M., & Hensgen, D. (2000). *Task execution time modeling for heterogeneous computing systems.* Cancún.

Bagley, J. (1967). The behaviour of adaptive systems which employ genetic and correlation algorithms. *Dissertation Abstracst International 28* , 12.

BeInGRID. (2008). *Gridipedia*. Retrieved Septiembre 1, 2009, from http://www.gridipedia.eu

Braun, H., Siegel, T., Beck, N., Bölöni, L., Maheswaran, M., Reuther, A., y otros. (2001). A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing* (61), 810–837.

Brisard, F., Drescher, M., Ly, A., McGough, S., Pulsipher, D., & Savva, A. (2005). *Job Submission Description Language Specification, versión 1.0.*

Cao, J., Spooner, D., Jarvis, S., Saini, S., & Nudd, G. (2004). Grid load balancing using intelligent agents. En *Future Generation Computer Systems special issue on Intelligent Grid Environments: Principles and Applications.*

Cingolani, P. (s.f.). *jFuzzyLogic*. Recuperado el 1 de Septiembre de 2009, de SourceForge.

Csaji, B., Monostori, L., & Kadar, B. (2004). Learning and Cooperation in a Distributed Market-Based Production Control System. En *Proceedings of the 5th International Workshop on Emergent Synthesis* (págs. 109–116).

Darwin, C. (1959). *On the Origin of Species by Means of Natural Selection.* John Murray.

EunJoung, B., SungJin, C., HongSoo, K., & ChongSun, H. (2008). *Advanced Job Scheduler Based on Markov Availability Model and Resource Selection in Desktop Grid Computing Environment.* Heidelberg: Springer.

Foster, I. (2005). *A Globus Tookit Premier.*

Goldberg, D. (1898). *Genetic algorithms in search, optimization and machine learning.* Adisson-Wesley.

Greene, W. (2001). Dynamic Load-Balancing via a Genetic Algorithm. Dallas: 13th IEEE International Conference on Tools with Artificial Intelligence.

Holland, J. (1962). Information processing in adaptive systems. *Information Processing in the Nervous Systems, Proceedings of the National Union of Psysiological Sciences, 3*, págs. 330-339.

Huang, W., French, T., Maple, C., & Bessis, N. (2006). Can Intelligent Optimisation Techniques Improve Computing Job Scheduling In A Grid Environment. *Problem and Proposal UK e-Science* .

Huedo, E., Montero, R., & Llorente, I. (2005). The GridWay Framework for Adaptive Scheduling and Execution on Grids. *Scalable Computing - Practice and Experience , 6* (3), págs. 1-8.

I. Foster, C. K. (1998). The Globus Project: A Status Report. *Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop*, (pp. 4-18).

Iordache, G., Boboila, M., Pop, F., & Stratan, C. (2008). *Descentralized Grid Scheduling Using Genetic Algorithms.* Heidelberg: Springer.

Java Object Oriented Neural Engine. (s.f.). Recuperado el 1 de Septiembre de 2009, de SourceForge: http://sourceforge.net/projects/joone/index.pdf

Jia, Y., Rajkumar, B., & Kotagiri, R. (2008). *Workflow Scheduling Algorithms for Grid Computing.* Heidelberg: Springer.

Kamer, K., Bora, U., & Cevdet, A. (2008). *Adapting Iterative-Improvement Heuristics for Scheduling File-Sharing Tasks on Heterogeneous Platforms.* Heidelberg: Springer.

Kannan, S., Roberts, M., Mayes, P., Brelsford, D., & Skovira, J. (2001). *Workload Management with LoadLeveler.* IBM Redbook publication.

LaTorre, A., Peña, J., Robles, V., & de Miguel, P. (2008). *Supercomputer Scheduling with Combined Evolutionary Techniques.* Heidelberg: Springer.

Lay, G. (2003). *Real-time scheduling.*

Mahmood, A. (2000). A Hybrid Genetic Algorithm for Task Scheduling in Multiprocessor Real-Time Systems. *Journal of Studies in Informatics and Control , 9* (3).

Moreno Velo, F. J., Baturone, I., Sánchez Solano, S., & Barriga, A. (2001). XFuzzy 3.0: A Development Environment for Fuzzy Systems. *International Conference in Fuzzy Logic and Technology (EUSFLAT 2001)*, (págs. 93-96). Leicester.

Nadeem, F., Prodan, R., Thomas, F., & Iosup, A. (2009). *A Framework for Resource Availability Characterization and On-Line Prediction in Large Scale Computational Grids.* Institute on Resource, Management and Scheduling.

Page, A., & Naughton, T. (2005). Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing. En *Proceedings of the 19th International Parallel and Distributed Processing Symposium.* Denver.

Perez, J. K.-R. (2007). *Reinforcement learning for utility-based Grid scheduling.* Vancuver.

Pinedo, M. (2002). *Scheduling: Theory, Algorithms, and Systems.* Springer.

Prodan, R., & Fahringer, T. (2005). Dynamic scheduling of scientific workflow applications on the grid: a case study. Santa Fé: Symposium on Applied Computing.

Ramamritham, K. (1993). Allocation and scheduling of precedence related periodic tasks. IEEE TPDS.

Schopf, J. M. (2006). *Ten Actions when Grid Scheduling.* Chicago.

Seredynski, F., Koronacki, J., & Janikow, C. (1999). Distributed Scheduling with Decomposed Optimization Criterion: Genetic Programming Approach. En *Rolim, J.D.P.* Heidelberg: Springer.

Sevarac, Z. (s.f.). *Neuroph - Java Neural Nerwork Framework.* Recuperado el 1 de Septiembre de 2009, de SourceForge: http://neuroph.sourceforge.net

Shimizu, Y., & Tanaka, Y. *Practical multi-objective scheduling through soft-computing approach.*

Sulistio, A., Cibej, U., Venugopal, S., Robic, B., & Buyya, R. (2008). A toolkit for modelling and simulating Data Grids: An extension to GridSim. En *Concurrency and Computation: Practice and Experience (CCPE)* (Wiley Press ed.). New York: Wiley Press.

Tarancón, S., Garmendia, L., & Santos, M. (2009). Red Neuronal Difusa para un Planificador de Entornos Grid. *Jornadas internacionales de Didáctica de las Matemáticas en Ingeniera. E.T.S.I. Caminos, UPM,* (págs. 313-320). Madrid.

Theys, M., Braun, T., Siegal, H., Maciejewski, A., & Kwok, Y. (2001). *Mapping Tasks onto Distributed Heterogeneous Computing Systems Using a Genetic Algorithm Approach.* Chichester: John Wiley.

Vengerov, D. (2005). Adaptive Utility-Based Scheduling in Resource-Constrained Systems. En S. J. Zhang, *AI 2005* (págs. 477-488). Heidelberg: Springer.

Wang, W., Yuan, C., & Liu, X. (2008). *A Fuzzy Approach to Multi-product Mixed Production Job Shop Scheduling Algorithm.* IEEE computer society.

Weichhart, G., Affenzeller, M., Reitbauer, A., & Wagner, S. (2004). Modelling of an Agent-Based Schedule Optimisation System. En *Proceedings of the IMS International Forum.*

Wikipedia. (2009). *Wikipedia*. Recuperado el 1 de Septiembre de 2009, de http://es.wikipedia.org/wiki/Computaci%C3%B3n_grid

Wu, A., Yu, H., Jin, S., Lin, K.-C., & Schiavone, G. (2004). An incremental genetic algorithm approach to multiprocessor scheduling. En *IEEE Trans. on Parallel and Distributed Systems 15 (9)* (págs. 824–834).

Xhafa, F., & Abraham, A. (2008). *Meta-heuristics for Grid Scheduling Problems.* Berlin Heidelberg: Springer-Verlag Berlin Heidelberg.

Xhafa, F., Alba, E., Dorronsoro, B., Duran, B., & Abraham, A. (2008). *Efficient Batch Job Scheduling in Grids Using Cellular Memetic Algorithms.* Heidelberg: Springer.

Yoo, A., Jette, M., & Grondona, M. (2003). SLURM: Simple Linux Utility for Resource Management. En *Lecture Notes in Computer Science* (Vol. 2862, págs. 44-60). Springer-Verlag.

Yu, K.-M., Luo, Z.-J., Chou, C.-H., Chen, C.-K., & Zhou, J. (2007). *A Fuzzy Neural Network Based Schdeuling Algorithm for Job Assignment on Computational Grids.* Berlin / Heidelberg: Springer Berlin / Heidelberg.

Zadeh, L. (1965). Fuzzy Sets. *Information and Control* (8), págs. 338-353.

Zomaya, A., Ward, C., & Macey, B. Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues.

RENUNCIA

El abajo firmante, matriculado en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "SQS: Supercomputing Quality Scheduling: A Soft-computing Approach", realizado durante el curso académico 2008-2009 bajo la dirección de Luis Garmendia Salvador y con la colaboración externa de dirección de Matilde Santos Peñas en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Sergio Tarancón Faus